Universally Composable Adaptor Signatures

Paul Gerhart[1], Daniel Rausch[2], and Dominique Schröder[1]

[1] TU Wien, Austria
[2] University of Stuttgart, Germany

Abstract. Adaptor signatures extend the functionality of digital signatures by enabling the computation of *pre-signatures* on messages relative to statements in NP relations. Pre-signatures are publicly verifiable objects that simultaneously hide and commit to a standard signature on the same message. Anyone possessing a valid witness for the statement can adapt the pre-signature into a full signature under the underlying signature scheme. Since adaptor signatures are commonly used as building blocks in larger systems, in particular blockchain protocols, it is natural to seek a security definition in the Universal Composability (UC) framework. Tairi et al. (CCS'23) recently took a first step in this direction by proposing a UC functionality for adaptor signatures.

This paper investigates UC-secure adaptor signatures as a primitive, making both negative and positive contributions. On the negative side, we show that the functionality proposed by Tairi et al. suffers from critical limitations:

- The functionality fails to guarantee *extractability* and *adaptability*, which are core security properties of adaptor signatures, to higher-level protocols.
- No adaptor signature scheme can realize the functionality.

On the positive side, we propose a new UC functionality that faithfully captures the latest security guarantees of adaptor signatures as formalized via game-based notions by Gerhart et al. (EUROCRYPT'24).

- Our functionality guarantees extractability and pre-signature adaptability in a way that is composable and meaningful for higher-level protocols.
- We show that it is realizable by an enhanced Schnorr-based adaptor signature scheme that we construct. Our construction maintains compatibility with existing infrastructure and is efficient enough for practical deployment, particularly in Bitcoin-like environments.

Keywords: Adaptor Signatures, Universal Composability, Schnorr Adaptor Signatures

1 Introduction

Smart contracts are a foundational component of decentralized finance (DeFi), enabling automated and trustless interactions between mutually distrusting parties by delegating enforcement to the blockchain and thereby circumventing the classical impossibility of two-party fairness [Cle86]. Their expressive power has fueled widespread adoption across DeFi protocols, NFT platforms, and token swaps. As of 2026, over $118 billion is locked in smart contracts [DeF25], with more than 77 million contracts deployed on Ethereum alone [Eth25]. Despite their success, smart contracts suffer from inherent limitations [VST24]. Execution costs are high, particularly on Ethereum, where gas fees make even simple operations expensive [SHB+22]. Moreover, reliance on expressive scripting languages prevents deployment on blockchains such as Bitcoin, which intentionally restrict on-chain programmability [Poe17].

Adaptor signatures. To address these challenges, cryptographers introduced *adaptor signatures* [AEE+21], a primitive that enables fair exchange on blockchains without the need for complex on-chain scripting. Adaptor signatures extend standard digital signatures by enabling the computation of *pre-signatures* for NP statements. A pre-signature $\widetilde{\sigma}$ is a publicly verifiable object that commits to a valid signature σ on a message m and a statement Y, while hiding the signature. Anyone possessing the corresponding witness y for Y can transform $\widetilde{\sigma}$ into the valid signature σ. Most importantly, each party possessing both the pre-signature $\widetilde{\sigma}$ and the adapted signature σ can extract a valid witness for Y.

When deployed in blockchain-based protocols, adaptor signatures enable a fair exchange of a payment for a witness. Publishing an adapted signature authorizes a payment on the ledger while simultaneously revealing the witness. This coupling ensures that obtaining the payment and releasing the witness occur atomically, yielding fairness for both parties. While this fair exchange of a payment for a witness appears highly limited in expressiveness, protocols based on adaptor signatures have proven to be remarkably versatile. Beyond realizing conditional payments and atomic swaps without on-chain scripting [AEE+21,EFH+21], adaptor signatures also enable the construction of payment channels [MMS+19,DW15,MBB+19,EFHR20], privacy-preserving coin mixing [GMM+22,QPM+23], oracle-based payments [MTV+23], fair digital signature exchange [HPDB25,TMM22], and puzzle-solving incentives [VST24]. A major driver of their adoption is compatibility with existing signature standards. In particular, Bitcoin's Taproot upgrade [WNR20] enables native deployment of Schnorr-based adaptor signatures, allowing these protocols to be run on Bitcoin.

Adaptor signatures in higher-level protocols A common feature shared by most protocols based on adaptor signatures is that adaptor signatures are not used as a standalone primitive. Instead, they are combined with other cryptographic building blocks, such as digital signatures, encryption schemes, zero-knowledge proofs, and commitment mechanisms, and are deployed in environments that commonly include a blockchain or ledger [GMM+22,TMS23,QPM+23]. Reasoning about the security of such protocols by directly reducing each security claim to the underlying assumptions of all involved primitives is highly involved, requiring bespoke and tightly coupled security proofs that are specific to and have to be redone for each construction. This is laborious, repetitive, increases the risk for errors, and can even become impractical for complex protocols consisting of many building blocks interacting with each other. For this reason, these protocols are commonly analyzed in the Universal Composability framework (UC, e.g., [Can20,KTR20b,CKKR19]), where individual building blocks can be modeled as ideal functionalities that are composed to form more complex protocols.

Such UC-secure building blocks, formalized via ideal functionalities, provide security in arbitrary contexts, support modular analyses, and allow security results to be reused across protocols without redoing proofs. Security can be established once for a primitive in isolation by showing that it realizes a suitable ideal functionality, generally by performing a reduction to the underlying assumptions. A powerful composition theorem then directly implies that *every* more complex protocol built from such an ideal functionality remains secure if the functionality is replaced by *any* of its realizations – without having to perform another tedious and error-prone reduction that would now have to take the entirety of the more complex protocol into account. It is hence standard practice to study and formalize cryptographic primitives within the UC framework that can then be used as modular building blocks for other protocols, e.g., [Can04,KTR20a,CF01,CKS11,KR17,GMY06,KT09,BH04,FHH14,HM04].

The missing piece. While most cryptographic building blocks used in adaptor-signature-based protocols are available as UC ideal functionalities, for a long time adaptor signatures themselves lacked such a definition. This results in an unfortunate situation. Even if a higher-level protocol is analyzed in a UC model where all other components are idealized, the analysis must still fall back to reducing the entire protocol to security assumptions of adaptor signatures as defined via cryptographic security games. Every new protocol, hence, requires involved custom reductions that intertwine the game-based security of adaptor signatures with the UC analysis of the surrounding protocol [GMM+22,TMM22,HLTW24]. This voids much of the benefit of performing a UC analysis in the first place by undermining modularity, complicating security proofs, and obscuring the guarantees obtained by higher-level applications.

At CCS 2023, Tairi et al. [TMS23] took an important step towards resolving this gap by proposing the first and only ideal functionality for adaptor signatures, which we denote by $\mathcal{F}_{[\text{TMS23}]}$. This proposal raised the hope that adaptor signatures can finally be treated as a fully composable building block and integrated seamlessly into UC protocols. Unfortunately, as we show as one of our main results in this paper, this is not yet the case due to several critical issues of $\mathcal{F}_{[\text{TMS23}]}$.

1.1 Our Contributions

The variety of gaps within the formalization of $\mathcal{F}_{[\text{TMS23}]}$ illustrate that the design of an ideal functionality for adaptor signatures is much more difficult than perhaps initially expected, and it remains an open question

whether UC-secure adaptor signature schemes even exist in the first place. In this work we therefore perform a *foundational study of adaptor signature schemes as a primitive within the UC framework*, establishing both negative and positive results alongside novel conceptual insights.

Negative Results. We demonstrate several fundamental limitations of the functionality $\mathcal{F}_{[\text{TMS23}]}$ proposed by Tairi et al. [TMS23]:

- We prove that no adaptor signature scheme can realize $\mathcal{F}_{[\text{TMS23}]}$.
- Even if a realizing scheme were to exist, we show that $\mathcal{F}_{[\text{TMS23}]}$ fails to formalize extractability and adaptability, which together constitute the defining security properties of adaptor signatures. As a result, the functionality does not provide meaningful compositional guarantees to higher-level protocols.

Conceptual Insights. Motivated by these negative results, we investigate whether adaptor signatures can be captured in the UC framework and, if so, which structural properties are inherently required for UC security, thus obtaining fundamental insights into this primitive.

As one of our key insights, our analysis reveals a previously unknown gap between established game-based security definitions for adaptor signatures [GSST24] and fully composable security. In particular, we identify an additional requirement that is not implied by existing game-based notions but turns out to be essential in the UC setting.

Positive Results. Building on these insights, we obtain the following positive results:

- We propose a new ideal functionality for adaptor signatures that faithfully captures their core security guarantees while avoiding pitfalls of the previous functionality.
- We prove that any adaptor signature scheme that satisfies the established game-based security notions of [GSST24] and additionally fulfills the new requirement we identify realizes our ideal functionality.
- We show that the standard Schnorr-based adaptor signature construction, while secure under the notions of [GSST24], does not satisfy this additional requirement.
- We construct an enhanced Schnorr-based adaptor signature scheme that satisfies all core security guarantees as well as the additional requirement, yielding the first UC-secure adaptor signature scheme. Our construction introduces no new assumptions, remains highly efficient, and preserves compatibility with Schnorr signatures, making it suitable for deployment on Bitcoin and similar platforms.

Crucially, our positive results finally allow protocol designers to treat adaptor signatures seamlessly as a fully composable modular building block within their UC analyses.

1.2 Related Work

Adaptor signatures were initially introduced in the informal work of Poelstra [Poe17] and later received a formal treatment by Aumayr et al. [AEE+21]. Dai et al.[DOY22] identified shortcomings in this formalization and proposed refined game-based security definitions. Gerhart et al. [GSST24] extended this refined framework and provided game-based security definitions for adaptor signatures in a broader context, including coin-mixing services. These further extensions were necessary, since adaptor signatures have been applied in a variety of contexts, private coin mixing [GMM+22,QPM+23], oracle-based payment systems [MTV+23], and fair signature exchange [HPDB25,TMM22] beyond the initial application for Payment Channel Networks (PCNs) [MMS+19,DW15,MBB+19,EFHR20].

Other lines of work for adaptor signatures have focused on identifying adaptor signatures under various assumptions. Adaptor signatures have been found in the post-quantum setting, including proposals by Esgin et al.[EEE20], Tairi et al.[TMM20], and Renan et al.[RK24]. A generic construction for adaptor signatures exists from multi-party [CLTZ24] computation, and a class of dichotomic signatures can be generally transformed into adaptor signatures [GSST24,EFH+21].

To enable applications of adaptor signatures in which multiple signers are required, threshold and two-party adaptor signatures have been introduced [BGKS25,KOT24,EFH+21]. To extend the general extracting functionality, Vanjani et al. [VST24] introduced functional adaptor signatures, in which the signer extracts a

function of the witness rather than the witness itself. A treatment for adaptor signatures in the Universal Composability (UC) framework was developed by Tairi, Moreno-Sanchez, and Schneidewind [TMS23]. We provide a detailed comparison of our approach with that in Section 3.2. In addition to the work directly related to adaptor signatures, adaptor signatures share properties with verifiably encrypted signatures [BGLS03,RS09]. However, these schemes differ fundamentally in that they do not support public adaptivity or efficient witness extraction from the signature–ciphertext pair.

2 Preliminaries

In this section, we establish our notation, the foundations of adaptor signatures, and the UC framework. We recall definitions for non-interactive zero-knowledge proofs, signatures, and hard relations in Appendix A.

Notation By $x \leftarrow\!\$\ X$, we denote the uniform sampling of x from the set X, and λ represents the security parameter. By $y \leftarrow \mathsf{A}(x;r)$ we denote a probabilistic polynomial time (PPT) algorithm A that on input x and randomness r, outputs y. When A is a deterministic polynomial time (DPT) algorithm, we use the notation $x := \mathsf{A}(y)$. When we write "*if* $\exists x$" in an algorithm to check the existence of a value (e.g., in a tuple or relation), then within that if clause we bind the variable x to refer to that value. If there are multiple such values, x refers to the chronologically first one. We call a function $\mathsf{negl} : \mathbb{N} \to \mathbb{R}$ negligible in n if for every $k \in \mathbb{N}$, there exists $\lambda_0 \in \mathbb{N}$ s.t. for every $\lambda \geq \lambda_0$ it holds that $|\mathsf{negl}(\lambda)| \leq \lambda^{-k}$. We use the function **assert** to check if a condition holds. The function call **assert** condition aborts the game in which it is called, if condition evaluates to false.

The UC framework Universal Composability (UC) is a simulation-based paradigm for modeling and proving security of protocols (e.g., [Can20,KTR20b,CKKR19]). In a UC security analysis, one first defines an *ideal functionality* $\mathcal{F}$ that specifies the intended security and functionality of a protocol. The functionality $\mathcal{F}$ is secure by definition, but typically cannot be run in practice. To analyze the security of a concrete protocol $\mathcal{P}$ (called the *real protocol*), one shows that there exists a probabilistic polynomial-time (PPT) *simulator* $\mathcal{S}$ that controls the adversarial interface of $\mathcal{F}$ such that no PPT *environment* $\mathcal{E}$ can distinguish whether it is interacting with the real world (running $\mathcal{P}$) or the ideal world (running $\mathcal{F}$ with $\mathcal{S}$, denoted by $\mathcal{F} \mid \mathcal{S}$). Since $\mathcal{F}$ is secure by definition, this indistinguishability implies that $\mathcal{P}$ preserves the same guarantees. In this case, we say that $\mathcal{P}$ securely implements, or *(UC-)realizes*, $\mathcal{F}$, written as $\mathcal{P} \leq \mathcal{F}$. The UC framework supports a powerful *composition theorem*: if $\mathcal{P} \leq \mathcal{F}$, then any PPT protocol $\mathcal{Q}$ using $\mathcal{P}$ as a subroutine (written $\mathcal{Q} \mid \mathcal{P}$) also realizes the version using $\mathcal{F}$ instead (i.e., $\mathcal{Q} \mid \mathcal{P} \leq \mathcal{Q} \mid \mathcal{F}$). This allows one to first prove that $\mathcal{Q} \mid \mathcal{F}$ realizes another ideal functionality $\mathcal{F}'$, and then replace $\mathcal{F}$ with $\mathcal{P}$ without requiring further proof. Protocols such as $\mathcal{Q} \mid \mathcal{F}$ that use ideal subroutines are called hybrid protocols.

Adaptor signatures Adaptor signatures are primarily used in cryptocurrency protocols, such as atomic swaps [AEE+21] and payment channel constructions [EFH+21], where signatures must conform to a fixed, predefined format (e.g., Schnorr or ECDSA) to be accepted by the blockchain. This practical requirement means that adaptor signatures must produce valid instances of the underlying scheme. This limits the design space and makes compatibility with existing formats a critical constraint. Therefore, an *adaptor signature scheme* $\mathsf{AS}_{\Sigma,\mathcal{R}}$ extends a signature scheme $\Sigma = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vrfy})$ with four additional algorithms: $\mathsf{AS}_{\Sigma,\mathcal{R}} = (\mathsf{pSign}, \mathsf{pVrfy}, \mathsf{Adapt}, \mathsf{Extract})$. The pre-signing algorithm $\mathsf{pSign}(\mathsf{sk}, x, Y)$ allows a signer with secret key sk to tie a statement $Y \in \mathcal{L}_{\mathcal{R}}$ and a message x, producing a pre-signature $\widetilde{\sigma}$. The pre-signature can be publicly verified using $\mathsf{pVrfy}(\mathsf{pk}, x, Y, \widetilde{\sigma})$ for a public key pk. Given a witness y for the statement Y, the adapting algorithm $\mathsf{Adapt}(\mathsf{pk}, \widetilde{\sigma}, y, Y)$ transforms the pre-signature into a valid full signature σ. Finally, the extracting algorithm $\mathsf{Extract}(\mathsf{pk}, \widetilde{\sigma}, \sigma, Y)$ recovers the witness y from a valid signature–pre-signature pair. We define adaptor signatures following the latest formulation in [GSST24]:

Definition 1 (Adaptor Signature). *An* adaptor signature scheme $\mathsf{AS}_{\Sigma,\mathcal{R}}$ *w.r.t. a signature scheme* $\Sigma = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vrfy})$ *and a hard relation* $\mathcal{R}$ *consists of a tuple of algorithms* $\mathsf{AS}_{\Sigma,\mathcal{R}} = (\mathsf{pSign}, \mathsf{Adapt}, \mathsf{pVrfy}, \mathsf{Extract})$ *defined as*

– $\widetilde{\sigma} \leftarrow \underline{\mathsf{pSign}(\mathsf{sk}, x, Y)}$. *The* pre-signing *algorithm is a probabilistic polynomial-time (PPT) algorithm that, on input a secret key* sk*, a message* $x \in \{0,1\}^{\ell_m}$*, and a statement* $Y \in \mathcal{L}_{\mathcal{R}}$*, outputs a pre-signature* $\widetilde{\sigma}$*.*
– $b \leftarrow \underline{\mathsf{pVrfy}(\mathsf{pk}, x, Y, \widetilde{\sigma})}$. *The* pre-verification *algorithm is a deterministic polynomial-time (DPT) algorithm that, on input a public key* pk*, message* $x \in \{0,1\}^{\ell_m}$*, statement* $Y \in \mathcal{L}_{\mathcal{R}}$*, and pre-signature* $\widetilde{\sigma}$*, outputs a bit* b *indicating acceptance or rejection.*
– $\sigma := \underline{\mathsf{Adapt}(\mathsf{pk}, \widetilde{\sigma}, y, Y)}$. *The* adapting *algorithm is a DPT algorithm that, given a public key* pk*, pre-signature* $\widetilde{\sigma}$*, witness* y*, and a statement* Y*, outputs a full signature* σ*.*
– $y := \underline{\mathsf{Extract}(\mathsf{pk}, \widetilde{\sigma}, \sigma, Y)}$. *The* extracting *algorithm is a DPT algorithm that, on input a public key* pk*, a pre-signature* $\widetilde{\sigma}$*, a full signature* σ*, and a statement* $Y \in \mathcal{L}_{\mathcal{R}}$*, outputs a witness* y *such that* $(Y, y) \in \mathcal{R}$*, or outputs* $\perp$ *if extraction fails.*

To improve readability, we often omit the subscript in $\mathsf{AS}_{\Sigma,\mathcal{R}}$ and simply write AS when the underlying signature scheme Σ and relation $\mathcal{R}$ are clear from context. For technical convenience, we require the Adapt algorithm to never output an error symbol. Additionally, we modify its input to explicitly include the statement Y with respect to which the pre-signature is being adapted. This change does not restrict generality: any language can be redefined so that the witness y includes the statement as part of its structure. However, making the statement explicit simplifies the formalization of adaptor signatures in the UC framework.

Beginning with applications in Payment Channel Networks (PCNs) [MMS+19,DW15,MBB+19,EFHR20], the security requirements for adaptor signatures have progressively evolved to accommodate more advanced use cases such as private coin mixing [GMM+22,QPM+23] and oracle-based payments [MTV+23], among others. Based on these applications, adaptor signatures are generally expected to provide the following two security properties [GSST24], where each property is formalized via a separate security game. We provide an intuition here and defer a more formal treatment to Appendix B.2.

Extractability. Extractability generalizes existential unforgeability for signatures to the adaptor signature setting by incorporating witness extraction: The adversary can query sign and pre-sign oracles to generate signatures on messages and pre-signatures on message–statement pairs. To break extractability, an adversary must output a valid signature on a message that was not signed before (unforgeability) and that also cannot be paired with any previously generated pre-signature to extract a valid witness for the corresponding statement (witness extraction). This property protects the *signer*: if a valid signature appears (e.g., on-chain), the signer is assured that it either produced a signature on the message itself or can extract a witness from it using a pre-signature it previously issued.

Pre-signature adaptability. Pre-signature adaptability ensures that any valid pre-signature, when combined with a correct witness, can always be transformed into a valid full signature on the corresponding message. In this sense, a pre-signature functions as a verifiable commitment to a future signature that the adapter can complete once a witness is obtained, even if the witness is not yet available at the time the pre-signature is received and verified. This property protects the adapter by guaranteeing that it is safe to proceed with an exchange whenever pre-signatures are valid, since eventual possession of a witness will necessarily yield a valid and publicly verifiable signature.

Schnorr adaptor signatures. Due to its compatibility with Bitcoin, the Schnorr adaptor signature [AEE+21] is the most prominent instance that we briefly recall and defer a formal description to Construction 2. The construction follows the structure of Schnorr signatures, which have a secret key $\mathsf{sk} \in \mathbb{Z}_q$ and a corresponding public key $\mathsf{pk} = g^{\mathsf{sk}}$, and uses the discrete logarithm (DLog) relation over a prime-order group $\mathbb{G}$. A statement $Y \in \mathbb{G}$ is of the form $Y = g^y$ for a witness $y \in \mathbb{Z}_q$. To produce a pre-signature on a message x, the signer samples a random $r \leftarrow\!\!\$\, \mathbb{Z}_p$, computes the nonce $R = g^r$, and derives the challenge $c = \mathsf{H}_{\mathsf{Sign}}(\mathsf{pk}, x, R \cdot Y)$. The signer then computes $s = r + c \cdot \mathsf{sk}$ and returns the resulting pre-signature $\widetilde{\sigma} = (R \cdot Y, s)$. Given a valid witness y for Y, the adaptor can compute a full signature $\sigma = (R \cdot Y, s + y)$, which is a valid Schnorr signature under the nonce $R \cdot Y = g^{r+y}$. Anyone who receives both the pre-signature $\widetilde{\sigma}$ and the full signature σ can extract the embedded witness as $y = s' - s$ where s' is the second component of σ. A pre-signature $\widetilde{\sigma} = (R', s)$ is considered *valid* for a message x and a statement Y if, given the public key pk, the verifier can compute

the challenge $c = \mathsf{H}_{\mathsf{Sign}}(\mathsf{pk}, x, R')$ and check that the following equality holds: $g^s \cdot Y \stackrel{?}{=} R' \cdot \mathsf{pk}^c$. This condition ensures that the pre-signature is structurally consistent with the Schnorr signature equation and will yield a valid signature when adapted with a correct witness.

3 A First UC definition

As mentioned in the introduction, recently at CCS 2023 Tairi, Moreno-Sanchez, and Schneidewind proposed the first and so far only UC definition of adaptor signatures [TMS23]; we will refer to their functionality as $\mathcal{F}_{[\text{TMS23}]}$. Before we discuss $\mathcal{F}_{[\text{TMS23}]}$, we present two technical observations about $\mathcal{F}_{[\text{TMS23}]}$ that clarify its scope and place it in its timely cryptographic context.

First, Tairi et al. [TMS23] define adaptor signature security in terms of two earlier game-based notions: unforgeability and witness extractability. These notions have since been subsumed by the more expressive notion of *extractability*, which strictly generalizes both [DOY22]. While this shows that the UC formalization builds on a now-outdated foundation, this does not invalidate our analysis: all gaps we identify with respect to extractability remain meaningful, even when interpreted under the older notion of witness extractability. Moreover, the limitations of those earlier definitions, such as their one-time security scope [GSST24], were neither intended by [TMS23] nor enforced by $\mathcal{F}_{[\text{TMS23}]}$. For these reasons, we adopt extractability as the appropriate baseline for our evaluation, but our negative results apply regardless.

Second, [TMS23] models (pre-)signing as a distributed two-party protocol that jointly computes (pre-)signatures, as this was required for their case study. However, it is not just straightforward to interpret $\mathcal{F}_{[\text{TMS23}]}$ as a single-signer functionality but all of our results concerning $\mathcal{F}_{[\text{TMS23}]}$ are actually independent of whether a single- or two-signer setting is considered. For simplicity of presentation and since we aim for the standard case of single-signer UC-secure adaptor signatures, we present and discuss $\mathcal{F}_{[\text{TMS23}]}$ directly in the single-signer setting.

3.1 Definition of $\mathcal{F}_{[\text{TMS23}]}$

The definition by Tairi et al. builds on and extends the UC signature functionality of Kiayias et al. [KRDO17], which itself extends Canetti's foundational ideal functionality for digital signatures [Can04]. Roughly speaking, these prior ideal signature functionalities offer a Sign and a Vrfy oracle that the environment/parties in a higher-level protocol can access.[3]

When a party requests a signature on a message x, the functionality forwards the request to the simulator, which returns a signature σ to be handed back to the caller. The functionality internally maintains a list $\mathcal{Q}$ of all message–signature pairs it has issued in this way. Verification is likewise modeled as an interactive procedure. Before consulting the simulator on what the result of verification should be, the functionality first checks the list $\mathcal{Q}$ to enforce unforgeability and two functional properties:

- **(Existential) Unforgeability:** If the message x was never signed, i.e., no pair (x, σ') does appear in $\mathcal{Q}$, the functionality does not verify the signature.
- **Completeness/Correctness:** If the pair was honestly generated via signing, it verifies.
- **Consistency:** If the pair (x, σ) has been verified before, the functionality returns the same decision.

If none of these cases apply, the functionality defers to the simulators response. The final outcome is then stored in the list $\mathcal{Q}$ to ensure consistent future responses. Observe that these oracle definitions illustrate a fundamental design decision that Tairi et al. also largely follow: Rather than locally executing the corresponding algorithms to determine oracle outputs, they are instead obtained via an interactive process from the simulator.

[3] We say "oracle" when we want to refer to the interface provided by a UC protocol. We say "algorithm" to refer to the actual algorithm being used as part of a signature scheme. In the realization of an ideal functionality, an oracle internally just runs the corresponding algorithm and returns the result.

Augmenting the signature functionality for the adaptor setting. To obtain the ideal adaptor signature functionality $\mathcal{F}_{[\text{TMS23}]}$, Tairi et al. augment the UC signature functionality by introducing further oracles for pre-signing, pre-verification, adaptation, and extraction. To formalize additional security properties, they further introduce a second list $\mathcal{P}$ that records and keeps track of pre-signatures generated through honest pre-signing. We provide a high-level summary of $\mathcal{F}_{[\text{TMS23}]}$ in Figure 1 and briefly describe the main concepts next, which is sufficient to follow our results. We refer interested readers to Figure 9 in [TMS23] for the full formal definition.

When the pSign oracle is invoked on a message x and statement Y, $\mathcal{F}_{[\text{TMS23}]}$ queries the simulator to obtain a pre-signature $\widetilde{\sigma}$. This is subject to the restriction that pVrfy has not previously returned false for x and $\widetilde{\sigma}$. The functionality then stores a new entry in $\mathcal{P}$ of the form $e := (x, Y, \widetilde{\sigma}, 1)$, where the bit 1 marks the entry as valid.

The pVrfy oracle is defined similarly to the Vrfy oracle: Upon being called with a message x, a pre-signature $\widetilde{\sigma}$, and a statement Y, the oracle queries the simulator to determine the verification result, subject to the following restrictions which are computed based on the list $\mathcal{P}$: If $\widetilde{\sigma}$ was previously returned by the pSign oracle for x and Y, then it must verify (completeness/correctness). If the pSign oracle was never called on the message x and Y, then the verification must fail (existential unforgeability of pre-signatures). If this combination of inputs has been queried before to pVrfy, then the output must be the same (consistency). Finally, if the verification result has not yet been stored for a pre-signature–statement pair, then a new entry is added to the list $\mathcal{P}$, analogous to the pSign oracle.

The Adapt oracle can be called by providing a witness y and a pre-signature $\widetilde{\sigma}$. The oracle aborts if there is no unadapted entry $(x, Y, \widetilde{\sigma}, 1) \in \mathcal{P}$ containing $\widetilde{\sigma}$. It then checks that y is a witness for Y. If so, it locally runs the deterministic Adapt algorithm of the adaptor signature scheme on $\widetilde{\sigma}$ and y to compute a signature σ. It finally adds the pair (x, σ) to $\mathcal{Q}$ such that they will be verified by the Vrfy oracle and updates the entry in $\mathcal{P}$ to contain y and σ, effectively marking the pre-signature as adapted.

The Extract oracle is defined as the inverse of the Adapt oracle. That is, when called on a pre-signature $\widetilde{\sigma}$ and a signature σ, the Extract oracle checks whether there is a matching adapted pre-signature, such that for the pair $(\widetilde{\sigma}, \sigma)$, a witness y is stored. If so, the oracle returns y. In all other cases, the extraction oracle fails by returning $\perp$. Observe that the set $\mathcal{P}$ plays a central role in formalizing two of the major security properties from Section 2:

- **Pre-Signature Adaptability:** Any pre-signature either generated by pSign (which is guaranteed to pre-verify) or verified by pVrfy can be completed into a valid signature using a valid witness via a call to the Adapt oracle. Since the resulting signature is added to $\mathcal{Q}$, it will verify in the Vrfy oracle.
- **Extractability:** If a signature verifies under Vrfy, then it was added either by the Sign or the Adapt oracle to the set $\mathcal{Q}$. In the latter case, the Adapt oracle guarantees that this is only possible for a valid witness and a pre-signature on a message that was previously pre-signed. The definition of the Extract guarantees that the witness used for the call to Adapt will be returned when queried with that pre-signature.

Tairi et al. [TMS23] state that their ideal functionality can be realized by *any* adaptor signatures that satisfy standard game-based security:

Theorem 1 (Informal, Theorem 2 of [TMS23]). *Let $\mathsf{AS}_{\Sigma,\mathcal{R}}$ be an adaptor signature scheme for a hard relation $\mathcal{R}$ and secure signature scheme Σ. If $\mathsf{AS}_{\Sigma,\mathcal{R}}$ is extractable and pre-signature adaptable, then $\mathsf{AS}_{\Sigma,\mathcal{R}}$ UC-realizes the ideal functionality.*

Schnorr adaptor signatures, which are the most widely used adaptor signatures in practice, are known to achieve these game-based security notions [EFH+21,GSST24]. If the above theorem were correct, it would imply that Schnorr adaptor signatures realize the $\mathcal{F}_{[\text{TMS23}]}$ functionality. However, we show that the theorem is incorrect, and this implication does not hold.

3.2 Issues in the existing UC Definition

As our first main contribution, we identify a series of severe conceptual and technical issues in $\mathcal{F}_{[\text{TMS23}]}$. They can be categorized into three types depending on their implications:

The functionality maintains two lists: $\mathcal{Q}$ for signatures and $\mathcal{P}$ for (adapted) pre-signatures.

(Pre-)Signature Generation: Upon receiving a signing request for a message x, or a pre-signing request for a message–statement pair (x, Y), forward the request to Sim. Upon receiving a signature σ, abort if (x, σ) or (x, Y, σ) is marked invalid in $\mathcal{Q}$ or $\mathcal{P}$. Otherwise, store the result as valid and output σ.

- If this is a signature, record $(x, \sigma, 1)$ in $\mathcal{Q}$.
- If this is a pre-signature, record $(x, Y, \sigma, 1)$ in $\mathcal{P}$.

(Pre-)Signature Verification: Upon receiving a verification request for (x, Y, σ)—where $Y = \bot$ indicates that σ is a signature rather than a pre-signature—forward the request to Sim.

- If $(x, Y, \sigma, 1)$ is recorded in $\mathcal{Q}$ or $\mathcal{P}$, return 1 (completeness).
- If (x, Y, σ, f) exists, return the stored bit f (consistency).
- If no entry for $(x, Y, \cdot)$ exists, return 0 and record $(x, Y, \sigma, 0)$ in $\mathcal{Q}$ or $\mathcal{P}$ (unforgeability).
- Otherwise, return the simulator's decision and record (x, Y, σ, f) in $\mathcal{Q}$ or $\mathcal{P}$.

Adaptation: Upon receiving an adaptation request $(\mathsf{pk}, \widetilde{\sigma}, y)$, check whether $\widetilde{\sigma}$ is marked as valid and unadapted in $\mathcal{P}$, and whether pk is a valid public key. If not, abort. Otherwise, compute the adapted signature locally, update the entry in $\mathcal{P}$ with the adapted signature and witness.

Extraction: Upon receiving an extraction request $(\mathsf{pk}, \widetilde{\sigma}, \sigma)$, return the stored witness if $(\widetilde{\sigma}, \sigma)$ appears as an adapted pair in $\mathcal{P}$; otherwise, return $\bot$.

Fig. 1. Simplified functionality of Tairi et al. [TMS23]. We omit key generation, session identifiers, and internal bookkeeping details, which are not relevant for this overview. For the full specification, see [TMS23].

- We identify *additional requirements of $\mathcal{F}_{[TMS23]}$ on realizations that are not captured by Theorem 1*, therefore falsifying the theorem. Notably, we can show that Schnorr adaptor signatures *do not* realize $\mathcal{F}_{[\text{TMS23}]}$. This shows that stronger preconditions on realizations are needed.
- We show that $\mathcal{F}_{[TMS23]}$ *formalizes neither pre-signature adaptability nor extractability* towards higher-level protocols. A hybrid protocol relying on $\mathcal{F}_{[\text{TMS23}]}$ can hence not use these properties in its security analysis, defeating the point of a modular UC analysis.
- We show that $\mathcal{F}_{[TMS23]}$ *fundamentally cannot be realized by any adaptor signature scheme*, even under additional stronger assumptions. This is due to certain underlying design decisions that we discuss in the following.

In what follows, we explain each issue.

Requirement: existential pre-signature unforgeability. First, we observe that in $\mathcal{F}_{[\text{TMS23}]}$ a pre-signature may successfully pre-verify on a message-statement pair (x, Y) only if (x, Y) was previously pre-signed. We show two ways in which Schnorr adaptor signatures [AEE+21] do not comply with this notion.

The first way, we call "reverse adapting". Given a statement–witness pair (Y, y) and a valid Schnorr signature (σ_1, σ_2) on a message x, one can compute a valid pre-signature by "reversing" the adaptation: simply set $\widetilde{\sigma} = (\sigma_1, \sigma_2 - y)$. This pre-signature will satisfy the pre-verification check for (x, Y), even though it was never produced via an honest pre-signing invocation. In other words, an adversary with access to a valid signature and a statement-witness pair of its choice can forge a valid pre-signature on this statement.

The second way, we call "statement shifting". That is, given a valid pre-signature $\widetilde{\sigma} = (R \cdot Y, s)$ on some statement $Y = g^y$, it is possible to derive a valid pre-signature on a related statement without querying the pre-signing oracle. From a valid pre-signature $\widetilde{\sigma} = (R \cdot Y, s)$, one can compute a new valid pre-signature $\widetilde{\sigma}' = (R \cdot Y,\ s - 1)$ which is valid w.r.t. the shifted statement $Y' = Y \cdot g = g^{y+1}$. This transformation preserves the combined commitment $R \cdot Y$, which appears in the challenge computation. This pre-signature is valid under the same message as $\widetilde{\sigma}$ but under the shifted statement Y' and breaks pre-signature unforgeability. Interestingly, this property of Schnorr does not contradict extractability: For both $\widetilde{\sigma}$ and $\widetilde{\sigma}'$, adapting with the corresponding witness will yield the same overall signature σ since the addition of the witness removes

the difference introduced by statement shifting. Hence, when the signer obtains σ and runs extract on the original $\widetilde{\sigma}$ that he generated, he will obtain the witness y on his statement Y (rather than the $y+1$ on Y').

Requirement: extractability limited by the number of pre-signatures. Observe that the Extract oracle in $\mathcal{F}_{[\text{TMS23}]}$ returns a witness only if there exists an entry $e := (x, Y, \widetilde{\sigma}, 1, y, \sigma)$ that was previously added by the Adapt oracle by updating an entry of the form $e' := (x, Y, \widetilde{\sigma}, 1)$. Such an entry e' can only be added to $\mathcal{P}$ by the pSign and pVrfy oracles with at most one per call to each oracle. Hence $\mathcal{F}_{[\text{TMS23}]}$ requires something similar to a *one-to-one relationship* between the number of pre-signatures seen so far and witnesses that can be extracted. This is not implied by extractability and pre-signature adaptability and hence falsifies Theorem 1.

As a concrete counterexample, consider the adaptor signature scheme AS constructed in Figure 5 of [GSST24]. This scheme is extractable and pre-signature adaptable. But there exists a ppt attacker A that, given a single valid pre signature $\widetilde{\sigma}$, can compute two distinct full signatures $\sigma \neq \sigma'$ such that both signatures are valid and both of them extract w.r.t. $\widetilde{\sigma}$. This scheme therefore does not realize $\mathcal{F}_{[\text{TMS23}]}$: An environment can distinguish both cases by running A, who queries only the pSign oracle and only once, and then checking whether the Extract oracle returns a witness for both σ and σ'.

Requirement: adaptability/extraction only for verified pre-signatures. The Adapt and Extract oracles in $\mathcal{F}_{[\text{TMS23}]}$ will return an output only for pre-signatures that are marked as verified in $\mathcal{P}$, either because they were previously returned by the pSign oracle or successfully verified by the pVrfy oracle. No existing adaptor signature construction provides these properties, e.g., compare the Adapt (Extract) algorithm used for Schnorr adaptor signatures, which will also return an output for invalid pre-signatures. Since Adapt does not take the message x as input, existing algorithms also cannot simply be changed to first verify the pre-signature by running the pVrfy algorithm. While it might be possible to construct adaptor signatures with these properties, this really seems to be an unintended artifact of $\mathcal{F}_{[\text{TMS23}]}$.

Property not formalized: pre-signature adaptability. The Adapt oracle in $\mathcal{F}_{[\text{TMS23}]}$ uses the pre-signature input $\widetilde{\sigma}$ to locate *an* entry in the list that matches $\widetilde{\sigma}$, i.e, has the form $e := (x, Y, \widetilde{\sigma}, 1)$, and then updates that entry with an adapted signature σ. However, there might be a second entry $e' := (x', Y, \widetilde{\sigma}, 1)$, indicating that the same pre-signature is also associated with a different message x'. This can happen if the pSign oracle is called twice, once on x and once on x', and the attacker chooses to provide the same pre-signature $\widetilde{\sigma}$ for both queries.

As a consequence, the resulting adapted signature σ is guaranteed to be valid with respect to x but will be invalid for x', thereby *contradicting pre-signature adaptability.* Notably, this has the effect that a higher-level/hybrid protocol building on $\mathcal{F}_{[\text{TMS23}]}$ cannot actually use/assume that any pre-signature which verifies for some message x' will be adapted to a valid signature for the same message.

Property not formalized: extractability. Higher-level/hybrid protocols using $\mathcal{F}_{[\text{TMS23}]}$ further cannot make use of the extractability property since this is also not actually captured. Specifically, consider the following sequence: An honest signer P_{hon} in a higher-level protocol uses the pSign oracle on some message x to obtain a pre-signature $\widetilde{\sigma}$. A malicious party P_{mal} then uses the pVrfy oracle to verify a different pre-signature $\widetilde{\sigma}'$ w.r.t. the same message x. Since that message has already been pre-signed, $\mathcal{F}_{[\text{TMS23}]}$ allows the adversary, which is controlled by the environment in hybrid settings, to return successful verification for $\widetilde{\sigma}'$ (existential unforgeability for pre-signatures). The malicious party P_{mal} can then use the Adapt oracle on $\widetilde{\sigma}'$ to obtain a valid signature σ which $\mathcal{F}_{[\text{TMS23}]}$ then guarantees to extract (only) w.r.t. $\widetilde{\sigma}'$. However, σ *will not extract w.r.t. any pre-signature that* P_{hon} *knows*, i.e., malicious parties can obtain a valid signature without having to reveal a witness to the honest signer.

Jumping slightly ahead, it appears that the underlying issue lies in existential unforgeability actually being too weak to define extractability in a UC setting. We address this point by instead requiring a form of strong unforgeability. We will further discuss in Section 4.3 why this (or a similar) additional requirement appears to be inherently needed to bridge a previously unknown gap between established non-composable game-based security notions and fully composable UC security.

Unrealizability due to local algorithms. Fundamentally, the entire design of $\mathcal{F}_{\text{[TMS23]}}$ is based on the implicit assumption that all algorithms can be executed only by calling the corresponding oracle of $\mathcal{F}_{\text{[TMS23]}}$ such that the functionality always has full information about all inputs and actions. Such a design is possible for algorithms relying on secret inputs, such as the pSign algorithm, which requires a secret signing key that is abstracted by $\mathcal{F}_{\text{[TMS23]}}$ and hence can only be computed by using the corresponding pSign oracle of $\mathcal{F}_{\text{[TMS23]}}$. However, several algorithms, notably Adapt and Extract, take only public inputs and can hence be locally computed by an environment without using the oracles provided by $\mathcal{F}_{\text{[TMS23]}}$. This allows for several trivial distinguishing attacks on *all* adaptor signature schemes.

As just one concrete example, consider the following scenario: The environment uses the pSign oracle to obtain a (valid) pre-signature $\widetilde{\sigma}$ on a message x and a statement Y for which it knows a corresponding witness y. The environment then internally computes the Adapt *algorithm* on $(\widetilde{\sigma}, y)$ to obtain an adapted signature σ without querying the Adapt *oracle* of the functionality/real protocol. Finally, it calls the Extract oracle on $\widetilde{\sigma}$ and σ. By definition of $\mathcal{F}_{\text{[TMS23]}}$, since the Adapt oracle was never used, there is no record in $\mathcal{P}$ containing σ or even just the witness. Therefore, the Extract oracle will return $\perp$ in the ideal world. However, in the real world, the Extract oracle will run the Extract algorithm, which returns y as the signature was correctly adapted (formally follows from extractability, which is required to be able to realize $\mathcal{F}_{\text{[TMS23]}}$).

Unrealizability: one-time adaptability. A pre-signature $\widetilde{\sigma}$ in $\mathcal{F}_{\text{[TMS23]}}$ can only be adapted if an entry of the form $e := (x, Y, \widetilde{\sigma}, 1)$ is present in the list $\mathcal{P}$. Once the pre-signature is successfully adapted, the functionality updates the corresponding entry to $e := (x, Y, \widetilde{\sigma}, 1, y, \sigma)$. As a result, further invocations of the adaptation oracle using the same pre-signature are no longer possible, since no entry of the required form $(x, Y, \widetilde{\sigma}, 1)$ is present in $\mathcal{P}$.

This is impossible to achieve with any (non-interactive) adaptor signature scheme: the Adapt algorithm would have to fail for some party P_1 depending on whether a different party P_2 has already run the Adapt algorithm on the same input.

4 A New Functionality $\mathcal{F}_{\text{asig}}$

While $\mathcal{F}_{\text{[TMS23]}}$ is an important step towards universally composable adaptor signatures with strong security guarantees, the variety of technical issues we found in $\mathcal{F}_{\text{[TMS23]}}$ shows that defining such an ideal adaptor signature functionality turns out to be non-trivial and much harder than perhaps expected, if possible at all.

In this section, we will hence investigate and answer the following open question: *Is it possible to build an ideal functionality for universally composable adaptor signatures that formalizes the core properties of correctness, adaptability, and extractability towards higher-level protocols while also being realizable?* We answer this question affirmatively by presenting the first such functionality, denoted $\mathcal{F}_{\text{asig}}$. At the same time, we demonstrate that the definition of such an ideal functionality cannot reconcile adaptability and extractability without requiring an additional property that, at first glance, appears to be unrelated. This observation reveals a previously unknown gap between existing game-based security notions and fully composable UC security. We proceed by first providing a conceptual explanation of why reconciling adaptability and extractability is challenging within the UC framework, then constructing $\mathcal{F}_{\text{asig}}$, and finally discussing the implications of our functionality and the resulting insights for UC-secure adaptor signatures.

4.1 Conflicting Properties

A key challenge in defining *any* such ideal adaptor signature functionality lies in resolving the fundamental conflict between "positive" security properties, such as correctness and pre-signature adaptability, that imply that signatures must verify under certain conditions, and "negative" properties such as extractability, that imply that signatures must not verify under certain conditions. This conflict is often relatively easy to resolve in game-based notions, which consider only one property at a time and can argue over the entire run that has already concluded, while having access to all the information from that run. In contrast, a UC ideal functionality, which must guarantee all properties concurrently, encounters the issue that it must essentially

"commit" to one or the other decision, e.g., whether a signature verifies, while the run is still ongoing and with only limited information. Once committed, the functionality must be able to "patch" all future actions in such a way that they remain consistent with all required implications.

For example, consider pre-signature adaptability and extractability: If a pre-signature is accepted by the pVrfy oracle, pre-signature adaptability requires that any signature returned by the Adapt oracle verifies under the Vrfy oracle. At the same time, extractability requires that Vrfy must reject signatures if the message was neither signed nor the signature extracts w.r.t. a pre-signature previously produced by pSign. The ideal functionality now has to resolve the following situation: If it returns true for a pre-signature $\widetilde{\sigma}$ entered into the pVrfy oracle, then the functionality is committed in the sense that all signatures σ returned by the Adapt oracle must verify using the Vrfy oracle (pre-signature adaptability). At the same time, the ideal functionality should be defined to reject such signatures if they are not extractable. Furthermore, if an adapted σ is accepted by Vrfy due to pre-signature adaptability, then the ideal functionality is again committed in the sense that, for extractability, the Extract oracle has to return a witness for at least one pre-signature $\widetilde{\sigma}'$ previously returned by pSign that might potentially not even be the same as $\widetilde{\sigma}$, say, because pre-signatures are malleable.

Insight 1. Adaptor signatures have positive and negative security properties. While this does not directly cause issues in game-based definitions (but might conflict during a reduction), these properties can contradict each other in a UC functionality.

Insight 1 has several implications. Notably and unlike game-based definitions, $\mathcal{F}_{\text{asig}}$ additionally has to formalize and require from realizations another security property that, at first glance, might seem unrelated: *strongly unforgeable pre-signatures*. We discuss this requirement in Section 4.3 and give evidence that it (or a similar additional requirement) is inherently needed for any ideal adaptor signature functionality formalizing both adaptability and extractability.

4.2 A New Design Approach

The issues that we identified for $\mathcal{F}_{\text{[TMS23]}}$ are partly caused by an underlying fundamental design decision, namely, the implicit assumption that all algorithms can only and indeed are always executed via oracle calls. As explained above, this is not the case for locally executable algorithms relying only on publicly available information such as Adapt, Extract, pVrfy, Vrfy.

For $\mathcal{F}_{\text{asig}}$ we therefore start from scratch following a different design philosophy: Similar to other existing ideal signature functionalities such as [KT08,KTR20a], all oracles of $\mathcal{F}_{\text{asig}}$ will internally run the real algorithms and then only perform certain additional checks on top, possibly altering the output in the process, to enforce security properties. While many aspects and certain ideas of the functionality of Tairi et al. [TMS23] will also be represented in some form in $\mathcal{F}_{\text{asig}}$, by following this different definitional style, we can indeed show that $\mathcal{F}_{\text{asig}}$ is realizable while providing expected security guarantees to higher-level protocols.

Insight 2. Several algorithms of adaptor signatures are locally executable such that an environment can check whether oracles output the same as algorithms. To be realizable, an ideal functionality therefore must run the same algorithms except for cases that have negligible probability to occur (e.g., a forgery).

In what follows, we give an overview of the construction of $\mathcal{F}_{\text{asig}}$ via a sequence of refinement steps. We start from a baseline version that only runs the corresponding algorithms without any security guarantees, i.e., essentially the envisioned realization (cf. Figure 2). We then iteratively add changes that enforce security properties until we reach the final version (cf. Figure 3). This should not only help follow and verify the design rationale. It also serves as an extended proof sketch as these steps correspond to game hops in our realization proof(cf. Appendix C).

Correctness. We begin by modifying the Sign and pSign oracles to internally maintain lists $\mathsf{H}_{\mathsf{sign}}$ and $\mathsf{H}_{\mathsf{pSign}}$, which store all message-signature pairs or message-pre-signature-statement triples that have been successfully

The functionality is parameterized by a polytime decidable relation $\mathcal{R}$, ppt algorithms $\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{pSign}$ and dpt algorithms $\mathsf{Vrfy}, \mathsf{pVrfy}, \mathsf{Adapt}, \mathsf{Extract}$. Upon first activation, initialize this instance by running and storing $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$.

On $\mathtt{PubKey}$? from anyone, return $(\mathtt{PubKey}, \mathsf{pk})$.

On $(\mathtt{Sign}, x)$ for party pid in session $sid = (pid, sid')$, do:
Return $(\mathtt{Signature}, \mathsf{Sign}(\mathsf{sk}, x))$

On $(\mathtt{pSign}, x, Y)$ for party pid in session $sid = (pid, sid')$, do:
Return $(\mathtt{PreSignature}, \mathsf{pSign}(\mathsf{sk}, x, Y))$

On $(\mathtt{Adapt}, pk, \widetilde{\sigma}, y, Y)$ for party pid,
return $(\mathtt{Adapt}, \mathsf{Adapt}(pk, \widetilde{\sigma}, y, Y))$

On $(\mathtt{Extract}, pk, \widetilde{\sigma}, \sigma, Y)$ for party pid,
return $(\mathtt{Extract}, \mathsf{Extract}(pk, \widetilde{\sigma}, \sigma, Y))$

On $(\mathtt{pVerify}, pk, x, Y, \widetilde{\sigma})$ for party pid,
return $(\mathtt{VerResult}, \mathsf{pVrfy}(pk, x, Y, \widetilde{\sigma}))$

On $(\mathtt{Verify}, pk, x, \sigma)$ for party pid,
return $(\mathtt{VerResult}, \mathsf{Vrfy}(pk, x, \sigma))$

As usual, the attacker may corrupt any party pid dynamically and gains full control over that party. In addition, if pid is the owner of the key pair modeled by this instance, i.e., the session ID of this instance is $sid = (pid, sid')$, then the attacker also learns sk; for notational convenience, we set $\mathsf{corr} = \mathsf{true}$ iff this is the case.

Fig. 2. Baseline ideal functionality for adaptor signatures/envisioned realization. The functionality is parameterized by the local algorithms invoked in response to oracle queries. At this stage, no security properties are enforced; the functionality simply reflects the behavior of a concrete adaptor signature scheme when used as realization.

produced via signing or pre-signing, respectively. Additionally, we introduce the list $\mathsf{H}_{\mathsf{pVerify}}$, which records the same type of triples but for successfully verified calls to the pVrfy oracle. Finally, we add the list $\mathsf{H}_{\mathsf{adapt}}$, which stores tuples of the form $(\widetilde{\sigma}, Y, y, \sigma)$ whenever the Adapt oracle is used on input pre-signature $\widetilde{\sigma}$ and valid witness y for statement Y.

These lists are used as bookkeeping to enforce security properties in the following steps. We keep separate lists for each operation rather than trying to merge several steps into a single list, such as $\mathcal{P}$ in $\mathcal{F}_{[\text{TMS23}]}$. This makes it easier to formalize fine-grained security properties later on. It also reduces the risk of accidental incompatibilities with realizations due to locally executable algorithms: local execution only affects individual lists for the corresponding algorithms, rather than everything in a single merged list.

Based on these lists, we first enforce correctness. To this end, we augment the Vrfy and pVrfy oracles to consult the $\mathsf{H}_{\mathsf{sign}}$ and $\mathsf{H}_{\mathsf{pSign}}$ lists: When the oracle is called, it checks whether the inputs appear in $\mathsf{H}_{\mathsf{sign}}$ or $\mathsf{H}_{\mathsf{pSign}}$ respectively, i.e., were previously generated by a successful (pre-)sign operation. If so, the oracle overwrites a potentially negative verification result of the (pre-)verification algorithm with true, ensuring that all honestly generated (pre-)signatures always verify successfully. Additionally, we enhance the $\mathsf{Extract}$ oracle to preserve correctness in witness extraction for signatures computed by the Adapt oracle. More precisely, when extracting from a pre-signature–signature–statement triple $(\widetilde{\sigma}, \sigma, Y)$, the functionality checks whether this triple is recorded in $\mathsf{H}_{\mathsf{adapt}}$. If so, but the extract algorithm fails to return a valid witness for Y, the functionality overrides the output with the correct witness y previously stored in $\mathsf{H}_{\mathsf{adapt}}$ (choosing the first such witness if there are multiple stored in $\mathsf{H}_{\mathsf{adapt}}$). Enforcing extract correctness is not yet a very useful property on its own but will rather be needed to reconcile extractability with pre-signature adaptability in a later step (see paragraph *Pre-Signature Unforgeability*). We provide the corresponding version of our functionality in Figure 11, and formally prove in Claim 1 that every adaptor signature scheme AS satisfying both message-boundedness and correctness securely realizes this version of the functionality.

Pre-signature adaptability. Due to previous step, we can now add *pre-signature adaptability* via a very simple change: The Vrfy oracle is changed to check whether there exists the triple $(\widetilde{\sigma}, Y, y)$, such that the pre-signature $\widetilde{\sigma}$ is valid w.r.t. the message x and the statement Y, and the tuple $(\widetilde{\sigma}, Y, y, \sigma)$ is stored in $\mathsf{H}_{\mathsf{adapt}}$. If this exists, then the signature σ is marked as valid for x. This is sufficient to guarantee pre-signature adaptability, since a tuple is only added to $\mathsf{H}_{\mathsf{adapt}}$, if $(Y, y) \in \mathcal{R}$. While there are no guarantees in cases where signatures were adapted by running the Adapt algorithm locally rather than using the Adapt oracle, this guarantee is sufficient since honest parties in higher-level protocols will always use the oracle. We present the modifications for pre-signature adaptability in Figure 12 and show that it can be realized by adaptor signatures that additionally provide pre-signature adaptability in Claim 2. We will discuss consistency of our functionality in Section 4.3.

Pre-signature unforgeability. The next property we enforce is *pre-signature unforgeability*. As already mentioned in Section 3.2 and further discussed in Section 4.3, this property is not implied by the two main properties we want to model but is included to resolve an inherent tension between extractability and pre-signature adaptability in a composable ideal functionality. Concretely, we require that if a message x has not been signed, then it must be infeasible to produce a new valid pre-signature $\widetilde{\sigma}$ on x for some statement Y such that $\widetilde{\sigma}$ was not returned by running pSign on x and Y. On a technical level, this property is enforced by adding a check to the pVrfy oracle that overrides the output to false whenever a pre-signature would violate this unforgeability notion.

This condition resolves the aforementioned conflict: If x has already been signed, then extractability is trivially fulfilled, even if no witnesses can be computed by Extract, such that the functionality can safely allow pre-signature adaptability on any pre-signature for that message. Conversely, if x is unsigned, then any valid pre-signature that might get adapted must have originated from the pSign oracle, ensuring that the witness entered in the Adapt oracle (hence also in the list $\mathsf{H}_{\mathsf{adapt}}$) and hence returned by the Extract oracle will be extractable w.r.t. a pre-signature that the signer knows and for a statement Y that the signer chose. We defer a detailed discussion of this pre-signature unforgeability notion to Section 5 and provide the corresponding version of the functionality in Figure 13. We prove in Claim 3 that any adaptor signature scheme additionally satisfying our definition of pre-signature unforgeability (cf. Definition 2) realizes this functionality.

Extractability. The last guarantee we need to add is *extractability*. Due to careful preparations in the previous steps, which led to disjoint cases between the negative extractability and the positive pre-signature adaptability property, we can now formalize extractability via a simple change as follows. The Vrfy oracle is extended to additionally check the extractability property, but only in those cases where the signature is not already guaranteed to verify by correctness or by pre-signature adaptability. If the check determines that extractability would be violated, then the output of Vrfy is changed to be false.

This change does not break any of the previous properties while guaranteeing extractability for any signature that does not fall under the correctness or pre-signature adaptability check. For signatures that fall under the correctness check, we have that they were previously returned by the Sign oracle such that extractability always holds. For the case of signatures falling under pre-signature adaptability, extractability also always holds by what we discussed in the *pre-signature unforgeability* paragraph. Hence, the ideal functionality formalizes extractability for all possible cases.

We present the modifications for extractability in Figure 14. This finally yields our complete functionality $\mathcal{F}_{\text{asig}}$ as given in Figure 3. We show that this last modification is realizable by adaptor signature schemes that additionally provide extractability in Claim 4.

4.3 Discussion

In what follows, we discuss design and insights of $\mathcal{F}_{\text{asig}}$.

Design and modeling choices. In terms of party and session modeling, $\mathcal{F}_{\text{asig}}$ follows established conventions from previous ideal signature functionalities: Each instance of $\mathcal{F}_{\text{asig}}$ represents one signature key pair that can be used by an unbounded number of parties, each of them identified by a party ID pid_i. To distinguish

The functionality is parameterized by a polytime decidable relation $\mathcal{R}$, ppt algorithms $\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{pSign}$ and dpt algorithms $\mathsf{Vrfy}, \mathsf{pVrfy}, \mathsf{Adapt}, \mathsf{Extract}$. Upon first activation, initialize this instance by running and storing $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$.

On $\texttt{PubKey}$? from anyone, return $(\texttt{PubKey}, \mathsf{pk})$.

On $(\texttt{Sign}, x)$ for party pid in session $sid = (pid, sid')$, do:
- $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, x)$
- **if** $\sigma \neq \bot$**: add** σ **to** $\mathsf{H}_{\mathsf{sign}}[x]$
- Return $(\texttt{Signature}, \sigma)$.

On $(\texttt{pSign}, x, Y)$ for party pid in session $sid = (pid, sid')$, do:
- $\widetilde{\sigma} \leftarrow \mathsf{pSign}(\mathsf{sk}, x, Y)$
- **if** $\widetilde{\sigma} \neq \bot$**: add** $(\widetilde{\sigma}, Y)$ **to** $\mathsf{H}_{\mathsf{pSign}}[x]$
- Return $(\texttt{PreSignature}, \widetilde{\sigma})$

On $(\texttt{Adapt}, pk, \widetilde{\sigma}, y, Y)$ for party pid, do:
- $\sigma \leftarrow \mathsf{Adapt}(pk, \widetilde{\sigma}, y, Y)$
- **if** $pk = \mathsf{pk} \wedge (Y, y) \in \mathcal{R}$**: add** $(\widetilde{\sigma}, Y, y, \sigma)$ **to** $\mathsf{H}_{\mathsf{adapt}}$
- Return $(\texttt{Adapt}, \sigma)$

On $(\texttt{Extract}, pk, \widetilde{\sigma}, \sigma, Y)$ for party pid, do:
- $y := \mathsf{Extract}(pk, \widetilde{\sigma}, \sigma, Y)$
- **if** $(Y, y) \notin \mathcal{R} \wedge \exists (\widetilde{\sigma}, Y, y', \sigma) \in \mathsf{H}_{\mathsf{adapt}}$**:** $y := y'$ { *Extract Correctness*
- Return $(\texttt{Extract}, y)$

On $(\texttt{pVerify}, pk, x, Y, \widetilde{\sigma})$ for party pid, do:
- $b_{\mathsf{pVerify}} := \mathsf{pVrfy}(pk, x, Y, \widetilde{\sigma})$
- **if** $pk = \mathsf{pk} \wedge (\widetilde{\sigma}, Y) \in \mathsf{H}_{\mathsf{pSign}}[x]$**:** $b_{\mathsf{pVerify}} := 1$ { *Pre-Sign Correctness*
- **else if** $pk = \mathsf{pk} \wedge \neg\mathsf{corr} \wedge \mathsf{H}_{\mathsf{sign}}[x] = \emptyset$**:** $b_{\mathsf{pVerify}} := 0$ { *Pre-Sig Unforgeability*
- **if** $pk = \mathsf{pk} \wedge b_{\mathsf{pVerify}}$**: add** $(\widetilde{\sigma}, Y)$ $\mathsf{H}_{\mathsf{pVerify}}[x]$ **to**
- Return $(\texttt{VerResult}, b_{\mathsf{pVerify}})$

On $(\texttt{Verify}, pk, x, \sigma)$ for party pid, do:
- $b_{\mathsf{vrfy}} := \mathsf{Vrfy}(pk, x, \sigma)$
- $b_{\mathsf{ext}} := \exists (\widetilde{\sigma}, Y) \in \mathsf{H}_{\mathsf{pSign}}[x] : (Y, \mathsf{Extract}(pk, \widetilde{\sigma}, \sigma, Y)) \in \mathcal{R}$
- **if** $pk = \mathsf{pk} \wedge \in \mathsf{H}_{\mathsf{sign}}[x]$**:** $b_{\mathsf{vrfy}} := 1$ { *Correctness*
- **else if** $pk = \mathsf{pk} \wedge \exists (\widetilde{\sigma}, Y, y)$ s.t.
  $(\widetilde{\sigma}, Y) \in \mathsf{H}_{\mathsf{pVerify}}[x] \wedge (\widetilde{\sigma}, Y, y, \sigma) \in \mathsf{H}_{\mathsf{adapt}}$**:** $b_{\mathsf{vrfy}} := 1$ { *Pre-Sig Adaptability*
- **else if** $pk = \mathsf{pk} \wedge \neg\mathsf{corr} \wedge \mathsf{H}_{\mathsf{sign}}[x] = \emptyset \wedge \neg b_{\mathsf{ext}}$**:** $b_{\mathsf{vrfy}} := 0$ { *Extractability*
- Return $(\texttt{VerResult}, b_{\mathsf{vrfy}})$

As usual, the attacker may corrupt any party pid dynamically and gains full control over that party. In addition, if pid is the owner of the key pair modeled by this instance, i.e., the session ID of this instance is $sid = (pid, sid')$, then the attacker also learns sk; for notational convenience, we set $\mathsf{corr} = \mathsf{true}$ iff this is the case.

Fig. 3. Ideal adaptor signatures functionality $\mathcal{F}_{\text{asig}}$.

different key pairs, instances are identified via a session ID of the form $sid = (pid, sid')$ where pid is the ID of the party owning the key pair. The sub-identifier sid' allows for each party owning multiple key pairs, if needed. All parties pid_i can call all oracles of the functionality $\mathcal{F}_{\text{asig}}$ except for Sign and pSign, which are restricted to the party pid contained in the session ID $sid = (pid, sid')$. This captures that only the owner knows the secret key and hence is the only one able to (pre-)sign.

We chose to parameterize $\mathcal{F}_{\text{asig}}$ with a set of algorithms rather than, as is often done in such functionalities, asking the adversary to provide these algorithms at the start of the run. Note that this does not weaken the security statement or expressiveness. One can prove a higher-level protocol based on $\mathcal{F}_{\text{asig}}$ to be secure for any arbitrary but fixed set of parameters of $\mathcal{F}_{\text{asig}}$, which is precisely what one would have to show if $\mathcal{F}_{\text{asig}}$ were to instead ask the adversary to provide algorithms. However, fixing algorithms as parameters actually increases expressiveness: One can alternatively show the security of a higher-level protocol based on $\mathcal{F}_{\text{asig}}$ only for a specific set of algorithms, such as those compatible with the output format used in the higher-level protocol, rather than being forced to consider arbitrary ones. Fixing algorithms as parameters also completely avoids a class of artificial distinguishing attacks that often arise when an attacker chooses not to respond to a request for algorithms (see [CEK+16] for an in-depth discussion).

Properties provided towards higher-level protocols. Given that $\mathcal{F}_{\text{[TMS23]}}$ ended up formalizing neither adaptability nor extractability towards higher-level protocols, we should verify that $\mathcal{F}_{\text{asig}}$ as defined in Figure 3 indeed does: For adaptability this directly follows by definition of the Vrfy oracle when pre-verification and adaptation are performed by the oracles of $\mathcal{F}_{\text{asig}}$ before the resulting signature is verified. Note that this is sufficient since honest parties in higher-level protocols will be defined to always use these oracles of $\mathcal{F}_{\text{asig}}$ in the expected way. While malicious parties are of course free to (not) use the oracles of $\mathcal{F}_{\text{asig}}$ in arbitrary ways, including calling them in an unexpected order, $\mathcal{F}_{\text{asig}}$ does not, and indeed there is no need to provide security guarantees for such cases.

Extractability is slightly more involved to see. Consider the case where the Vrfy oracle returns true for some signature σ but the message x was not yet signed by Sign (extraction must work only in this cases). The *correctness* check will fail in that case, i.e., cannot be the cause for the true output. If the *adaptability* check returns false such that Vrfy reaches the *extractability* check, then extractability is enforced by definition. Finally, if the signature verifies due to the *adaptability* check returning true, then there is a pre-signature $\widetilde{\sigma}$ and statement Y such that (i) $(\widetilde{\sigma}, Y)$ was previously verified in pVrfy and (ii) the adapt oracle was called with a corresponding witness y. Due to strong pre-signature unforgeability and since x was not signed, (i) implies that $\widetilde{\sigma}$ was the result of a pSign oracle call on Y, i.e., the pre-signer knows this pre-signature and wants to obtain a witness for this statement. By definition of the Extract oracle, (ii) then implies that a valid witness will be returned when the pre-signer uses it on $(\widetilde{\sigma}, \sigma, Y)$ (albeit not necessarily the one used for adaptation).

Additional requirement: unforgeable pre-signatures. Recall from Section 4.1 that there is an inherent conflict between extractability and adaptability in ideal functionalities such as $\mathcal{F}_{\text{asig}}$. Without introducing additional requirements on pre-signatures, such as (strongly) unforgeable pre-signatures, the two notions would actually contradict each other within an ideal functionality. Intuitively, this is because extractability quantifies over all *honestly generated* pre-signatures, while adaptability considers all *pre-verified* pre-signatures, regardless of their origin.

To see this issue, consider the following example. Suppose an adversary generates a valid but fresh pre-signature $\widetilde{\sigma}^*$. For adaptability, an adapted signature σ resulting from $\widetilde{\sigma}^*$ must verify. However, extractability requires that the adapted signature is valid only if it also extracts with a pre-signature $\widetilde{\sigma}$ output by the pSign oracle. This cannot be enforced simultaneously by an ideal functionality without imposing further conditions on valid pre signatures. Crucially, while an ideal functionality would have obtained a witness for such a fresh pre-signature $\widetilde{\sigma}^*$ during adaptation, that witness might not match any statement that was previously pre-signed via the pSign oracle. This witness, hence, cannot be used to enforce extractability by programming the Extract oracle to cover cases where the (arbitrary, possibly attacker provided) Extract algorithm fails. The statement shifting property of Schnorr adaptor signatures that we identify in Section 3.2 actually gives a concrete example of this issue, where the witness y' used during adaptation does not match any previously

pre-signed statement Y and hence cannot be used by an ideal functionality to enforce extractability, yet this issue does not break game-based extractability.

Requiring (strong) pre-signature unforgeability is a natural solution which prevents this inconsistency by disallowing such adversarially generated pre-signatures altogether, not even by malling other pre-signatures. As a result, the set of pre-verifying pre-signatures coincides with the set of honestly generated pre-signatures such that any witness y obtained during adaptation can be used to guarantee extractability.

Insight 3. Extractability and adaptability generally contradict each other in ideal functionalities. We can resolve this contradiction by additionally requiring a notion of strongly unforgeable pre-signatures.

Fundamental limitation: consistency. $\mathcal{F}_{\text{asig}}$ does not guarantee consistency for, e.g., outputs of Vrfy. That is, given the same input signature, whether it verifies might change during the run. We first note that higher-level protocols generally do not require consistent verification responses such that this simplifies the definition without being an issue for applications of $\mathcal{F}_{\text{asig}}$. More fundamentally though, resolving the conflict of adaptability and extractability (cf. Insight 1) actually clashes with consistency in ideal adaptor signature functionalities, also beyond $\mathcal{F}_{\text{asig}}$.

To see this, consider a hybrid protocol execution in which a higher-level protocol obtains a pre-signature $\widetilde{\sigma}$ by invoking the pSign oracle of an ideal functionality with respect to some hard statement Y. The protocol then executes the deterministic Adapt algorithm *locally*, i.e., without using the oracle of the functionality, and obtains a signature σ. If the protocol queries the Vrfy oracle on σ, any definition of an ideal functionality has to enforce extractability, i.e., it may accept the signature only if it can return a witness when $\mathsf{Extract}$ is called on $(\widetilde{\sigma}, \sigma, Y)$. But the functionality never saw the witness as Adapt was executed locally, it cannot ask the adversary to provide one (in a hybrid world the adversary is controlled by the environment), and the $\mathsf{Extract}$ algorithm is arbitrary, possibly even attacker provided, such that it might return $\perp$. In such a case there is no choice for the functionality other than to reject σ. If this decision is then kept consistent for all future calls of Vrfy, the functionality will not provide adaptability: The higher-level protocol might next call the pVrfy and Adapt oracles with the same inputs, obtain the same signature σ, yet σ will remain invalid according to Vrfy.

Insight 4. Consistency is at odds with adaptability and extractability in ideal adaptor signature functionalities.

5 Realizing Our Functionality

As our main result, we show in Theorem 2 that any adaptor signature scheme satisfying the established game-based security notions and our additional requirement of *pre-signature unforgeability* UC-realizes $\mathcal{F}_{\text{asig}}$. We therefore begin by formally defining this additional security property.

Definition 2 (Strong Existentially Unforgeable Pre-Signatures). *An adaptor signature scheme* AS *satisfies* strong existentially unforgeable pre-signatures *if for every* PPT *adversary* $\mathcal{A}$ *there exists a negligible function* negl *such that for every* $\lambda \in \mathbb{N}$,

$$\Pr\left[\mathsf{Un\text{-}Forge\text{-}pSig}_{\mathcal{A},\mathsf{AS}}(\lambda) = 1\right] \leq \mathsf{negl}(\lambda),$$

where the experiment $\mathsf{Un\text{-}Forge\text{-}pSig}_{\mathcal{A},\mathsf{AS}}$ *is given in Figure 4, and the probability is over all randomness.*

We note that the previous functionality $\mathcal{F}_{[\text{TMS23}]}$ also formalizes a notion of existentially unforgeable pre-signatures. However, this notion is not strong enough to resolve the tension between adaptability and extractability since it allows for malleable pre-signatures (c.f. Section 3.2). In contrast, our notion enforces strongly existentially unforgeable pre-signatures, and we can show in Theorem 2 that this notion suffices to simultaneously enforce pre-signature adaptability and extractability. With strongly unforgeable pre-signatures, we are now ready to state our main theorem.

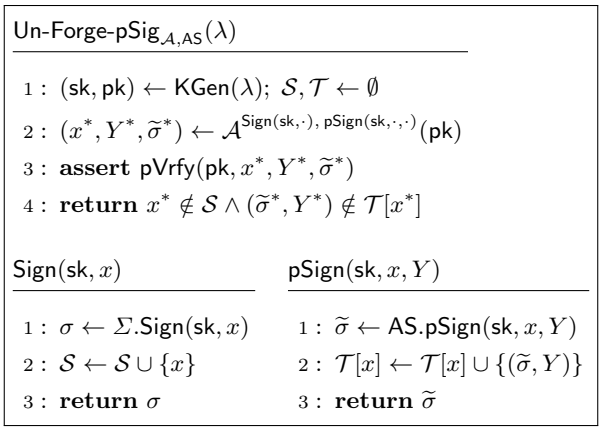

Fig. 4. The unforgeable pre-signature experiment $\mathsf{Un\text{-}Forge\text{-}pSig}_{\mathcal{A},\mathsf{AS}}(\lambda)$.

Theorem 2. *Let* $\mathsf{AS}_{\Sigma,\mathcal{R}} = (\mathsf{pSign}, \mathsf{pVrfy}, \mathsf{Adapt}, \mathsf{Extract})$ *be an adaptor signature scheme for a correct signature scheme* $\Sigma = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vf})$ *and a hard relation* $\mathcal{R}$*. Let* $\mathcal{F}_{asig}$ *be parameterized with the same algorithms. If* $\mathsf{AS}_{\Sigma,\mathcal{R}}$ *is extractable, pre-signature adaptable, correct, and has strong existentially unforgeable pre-signatures, then*

$$\mathsf{AS}_{\Sigma,\mathcal{R}} \leq \mathcal{F}_{asig}.$$

To show Theorem 2, we construct a simulator that ensures indistinguishability between the real and ideal worlds: no environment can distinguish whether it is interacting with the real-world protocol $\mathsf{AS}_{\Sigma,\mathcal{R}}$ or with the ideal functionality $\mathcal{F}_{\text{asig}}$ in interaction with the simulator. This demonstrates that the real protocol leaks no more information to an adversary than the ideal functionality, thereby achieving simulation-based security in the UC framework. The proof is structured as a sequence of games that follow along the modifications presented in Section 4. We defer the formal proof of Theorem 2 to Appendix C.

5.1 Schnorr Realization of $\mathcal{F}_{\text{asig}}$

We have already discussed the mechanics of Schnorr adaptor signatures in Section 2, and a formal construction is provided in Construction 2. In this section, we analyze whether Schnorr adaptor signatures can realize our ideal functionality $\mathcal{F}_{\text{asig}}$. We begin by showing that the standard construction fails to satisfy *pre-signature unforgeability*, and thus cannot realize $\mathcal{F}_{\text{asig}}$. To address this limitation, we introduce an enhanced construction that preserves the correctness and security guarantees of the original scheme, while additionally ensuring unforgeable pre-signatures. To the best of our knowledge, this yields the first adaptor signature scheme that UC-realizes an ideal functionality for adaptor signatures.

We begin by showing that the Schnorr adaptor signature scheme discussed in Section 2 does not realize $\mathcal{F}_{\text{asig}}$.

Lemma 1. *Schnorr adaptor signatures (Construction 2) do not satisfy pre-signature unforgeability (Definition 2).*

The proof of this lemma directly follows from the statement shifting property of Schnorr adaptor signatures (c.f. Section 3.2). For completeness, we provide a formal proof in Appendix E.

Pre-signature unforgeable schnorr adaptor signatures. To create an adaptor signature scheme with unforgeable pre-signatures, we propose a modification to Schnorr adaptor signatures in which each pre-signature $\widetilde{\sigma} = (R, \widetilde{s})$ on a message x is accompanied by a Schnorr proof-of-knowledge (PoK) of the underlying randomness r used in the nonce $R = g^r$. A statement for such a PoK is the tuple $(x, R, \widetilde{s}, Y)$ and the corresponding witness is r, such that $g^r = R$. By the simulation extractability of the PoK [BPW12], one cannot modify

the pre-signature to be valid w.r.t. a different Y without knowing the witness r. By the structure of Schnorr (pre-)signatures, requiring to prove knowledge of r is equivalent to requiring the knowledge of the signing key sk, since $\widetilde{s} = \mathsf{sk} \cdot h + r$ for a publicly known h. Thus, valid pre-signatures cannot be obtained by malling valid signatures or other valid pre-signatures without knowing the signing key sk.

This effectively enforces that only the legitimate signer can produce valid pre-signatures, rendering the scheme's pre-signatures strongly unforgeable. Most notably, we implement this proof of knowledge using a standard Schnorr-style proof of knowledge, which requires no additional setup beyond the existing random oracle and imposes the same computational overhead as signing itself.

In more detail, a pre-signature in our modified scheme consists of two components: the standard Schnorr adaptor pre-signature $\widetilde{\sigma}$, and a proof π proving knowledge of the randomness r used for computing $\widetilde{\sigma}$. To compute π, the signer samples a fresh scalar $r' \leftarrow\!\!\$\ \mathbb{Z}_p$ and computes $R' = g^{r'}$ along with the response $s' = r \cdot \mathsf{H}_{\mathsf{Prove}}(R', \widetilde{\sigma}, Y, x) + r'$, where $\mathsf{H}_{\mathsf{Prove}}$ is a hash function modeled as a random oracle. The full enhanced pre-signature is then

$$\widetilde{\sigma}' = (\widetilde{\sigma}, \pi) = \big(\widetilde{\sigma},\ (R', s')\big),$$

and it is considered valid if the pre-signature $\widetilde{\sigma}$ pre-verifies, *and* if π is valid. Checking the validity of π is similar to verifying Schnorr signatures. I.e., it requires verifying the equation

$$R^{\mathsf{H}_{\mathsf{Prove}}(R', \widetilde{\sigma}, Y, x)} \cdot R' \stackrel{?}{=} g^{s'}.$$

For the adapt and extract algorithms, our enhanced scheme ignores the additional proof and just runs the original algorithms on the pre-signature part. We note, that we indeed need a zero-knowledge proof and simply signing the pre-signature would not be sufficient, as during the security experiment, the adversary also has access to a signing oracle, and hence could compute such a signature on its own.

Construction 1 (Enhanced Schnorr Adaptor Signature Scheme) *Let* $\mathsf{AS} = (\mathsf{pSign}, \mathsf{Adapt}, \mathsf{pVrfy}, \mathsf{Extract})$ *denote the standard Schnorr adaptor signature scheme as defined in Figure 7. We define the enhanced scheme* $\mathsf{AS}' = (\mathsf{pSign}', \mathsf{Adapt}', \mathsf{pVrfy}', \mathsf{Extract}')$ *in Figure 5.*

$\mathsf{pSign}'(\mathsf{sk}, x, Y)$

1 : $r \leftarrow\!\!\$\ \mathbb{Z}_p$;
2 : $\widetilde{\sigma} := \mathsf{pSign}(\mathsf{sk}, x, Y; r)$
3 : $r' \leftarrow\!\!\$\ \mathbb{Z}_p;\ R' := g^{r'}$
4 : $c' := \mathsf{H}_{\mathsf{Prove}}(\mathsf{pk}, x, \widetilde{\sigma}, Y, R')$
5 : $s' := r' + c' \cdot r$
6 : **return** $(\widetilde{\sigma}, (R', s'))$

$\mathsf{pVrfy}'(\mathsf{pk}, x, Y, \widetilde{\sigma})$

1 : $((\widetilde{R}, \widetilde{s}), (R', s')) := \widetilde{\sigma}$
2 : $c' := \mathsf{H}_{\mathsf{Prove}}(\mathsf{pk}, x, \widetilde{\sigma}, Y, R')$
3 : $b_1 := \mathsf{pVrfy}(\mathsf{pk}, x, Y, \widetilde{\sigma}_1)$
4 : $b_2 := [g^{s'} = (\widetilde{R}/Y)^{c'} \cdot R']$
5 : **return** $b_1 \wedge b_2$

$\mathsf{Adapt}'(\mathsf{pk}, \widetilde{\sigma}, y, Y)$

1 : $((R \cdot Y, s), (\cdot, \cdot)) := \widetilde{\sigma}$
2 : **return** $(R \cdot Y, s + y)$

$\mathsf{Extract}'(\mathsf{pk}, \widetilde{\sigma}, \sigma, Y)$

1 : $(\cdot, \widetilde{s}), (\cdot, \cdot)) := \widetilde{\sigma}$
2 : $(\cdot, s) := \sigma$
3 : **return** $s - \widetilde{s}$

Fig. 5. Enhanced Schnorr Adaptor Signature Construction for the DLog relation.

Theorem 3. *The enhanced Schnorr adaptor signature construction (Construction 1) is correct and achieves extractability, pre-signature adaptability, and unforgeable pre-signatures in the random oracle model.*

We defer the proof of Theorem 3 to Appendix C, and provide intuition for why the theorem holds. For all desired properties except pre-signature unforgeability, we rely on the fact that the added proof of knowledge is simulation-extractable. This allows us to reduce the security of the enhanced scheme to that of the underlying Schnorr adaptor signature scheme. In particular, we can use the (pre-)signing oracles from the security experiments of the standard scheme to obtain ordinary pre-signatures, and then employ the simulator of the proof system to augment them into valid pre-signatures of the enhanced scheme, *without* knowing the underlying randomness. Using this approach, we demonstrate that all security properties of the standard Schnorr adaptor signature scheme are preserved in the enhanced construction.

For pre-signature unforgeability, we proceed similarly, but now rely on the *extractability* of the Schnorr adaptor signature scheme and the simulation-extractability of the PoK [BPW12]. We simulate the pre-signing and signing oracles using the oracles provided by the reduction against extractability and the simulator of the PoK. When the adversary outputs a pre-signature forgery, we use the extractor of the proof to extract the randomness used for such a pre-signature. Given this randomness, the reduction can compute the signing key and trivially break extractability, thereby contradicting the underlying scheme's security.

6 Additional Security Guarantees

Beyond the core security guarantees formalized in our functionality, depending on the specific application at hand, adaptor signatures are sometimes associated with varying additional properties. We discuss some of the most important ones in what follows.

Unique extractability [GSST24] This property ensures that each valid pre-signature can be used to extract a witness with respect to at most one valid signature. The resulting one-to-one relationship is required for some applications, such as private coin mixing [GMM+22].

In Appendix D we investigate the possibility of extending our baseline functionality $\mathcal{F}_{\text{asig}}$ to additionally provide unique extractability. To summarize our findings, we can answer this question affirmatively and provide a corresponding extended functionality $\mathcal{F}_{\text{asig}}^{(\text{ext})}$. Unfortunately, $\mathcal{F}_{\text{asig}}^{(\text{ext})}$ is significantly more complex as unique extractability needs to be enforced at every point that can introduce new valid (pre-)signatures, i.e., the pre-sign, sign, pre-verify, verify, and adapt oracles. We can show that any adaptor signature scheme satisfying corresponding game-based notions realizes $\mathcal{F}_{\text{asig}}^{(\text{ext})}$. We also show that the enhanced Schnorr adaptor signature scheme we proposed in Section 5.1 satisfies all requirements, i.e., also realizes the stronger functionality $\mathcal{F}_{\text{asig}}^{(\text{ext})}$ without requiring any further modifications.

Pre-verify soundness [GSST24] This property strengthens adaptability by additionally requiring that pre-signatures do not verify for invalid statements. Observe that, if the language cannot be decided in polynomial time without access to a witness, then *pre-verify soundness* cannot be defined in any ideal functionality, which is required to be PPT and is generally unaware of the witness at the time of pre-verification. In contrast, if the language is already efficiently decidable, as is typically the case for applications of adaptor signatures [BGKS25,MTV+23], then *pre-verify soundness* need not be enforced by an ideal functionality. The higher-level protocol can simply verify language membership, whenever required, on its own. For the sake of simplicity, we therefore opted not to include this property in our ideal functionality.

Unlinkability [DOY22,GSST24] This is a privacy property which ensures that, in the absence of the original pre-signature, an adversary cannot distinguish an adapted pre-signature from a standard signature. While it is likely possible, formalizing this privacy property in ideal functionality is non-trivial and, at the very least, complicates the definition significantly. Intuitively, this is because unlinkability holds only when the adversary is unaware of honestly computed pre-signatures. However, in UC the functionality provides all of its outputs to the environment, which acts as the adversary. Therefore, to enforce unlinkability, the functionality at least needs to include an additional mechanism to securely and secretly transmit pre-signatures between two honest parties without revealing them to the environment. This interactive mechanism must also be implemented in a realization, e.g., via a separate secure channel running alongside the adaptor signature

scheme. As unlinkability is also most often not required [TMS23], we opted for a simpler (non-interactive) ideal functionality without it and leave exploring this direction for future work.

Acknowledgements

This work is also partially supported by SBA Research (SBA-K1 NGC), which is a COMET Center within the COMET (Competence Centers for Excellent Technologies) Programme and funded by BMIMI, BMWET, and the federal state of Vienna. The COMET Programme is managed by FFG. This work is further partially supported by the Vienna Science and Technology Fund (WWTF) under grant ICT25-081. Paul Gerhart's research has been supported by the Google PhD Fellowship in Privacy, Safety, and Security.

References

AEE+21. Lukas Aumayr, Oguzhan Ersoy, Andreas Erwig, Sebastian Faust, Kristina Hostáková, Matteo Maffei, Pedro Moreno-Sanchez, and Siavash Riahi. Generalized channels from limited blockchain scripts and adaptor signatures. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part II*, volume 13091 of *LNCS*, pages 635–664. Springer, Cham, December 2021.

BCJZ21. Jacqueline Brendel, Cas Cremers, Dennis Jackson, and Mang Zhao. The provable security of Ed25519: Theory and practice. In *2021 IEEE Symposium on Security and Privacy*, pages 1659–1676. IEEE Computer Society Press, May 2021.

BGKS25. Ruben Baecker, Paul Gerhart, Jonathan Katz, and Dominique Schröder. Fair exchange for decentralized autonomous organizations via threshold adaptor signatures. Cryptology ePrint Archive, Report 2025/388, 2025.

BGLS03. Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In Eli Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 416–432. Springer, Berlin, Heidelberg, May 2003.

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, Berlin, Heidelberg, September 2004.

BN06. Mihir Bellare and Gregory Neven. Multi-signatures in the plain public-key model and a general forking lemma. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *ACM CCS 2006*, pages 390–399. ACM Press, October / November 2006.

BPW12. David Bernhard, Olivier Pereira, and Bogdan Warinschi. How not to prove yourself: Pitfalls of the Fiat-Shamir heuristic and applications to Helios. In Xiaoyun Wang and Kazue Sako, editors, *ASIACRYPT 2012*, volume 7658 of *LNCS*, pages 626–643. Springer, Berlin, Heidelberg, December 2012.

Can04. R. Canetti. Universally composable signature, certification, and authentication. In *Proceedings. 17th IEEE Computer Security Foundations Workshop, 2004.*, pages 219–233, 2004.

Can20. Ran Canetti. Universally composable security. *J. ACM*, 67(5), September 2020.

CDF+21. Cas Cremers, Samed Düzlü, Rune Fiedler, Marc Fischlin, and Christian Janson. BUFFing signature schemes beyond unforgeability and the case of post-quantum signatures. In *2021 IEEE Symposium on Security and Privacy*, pages 1696–1714. IEEE Computer Society Press, May 2021.

CEK+16. Jan Camenisch, Robert R. Enderlein, Stephan Krenn, Ralf Küsters, and Daniel Rausch. Universal composition with responsive environments. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part II*, volume 10032 of *LNCS*, pages 807–840. Springer, Berlin, Heidelberg, December 2016.

CF01. Ran Canetti and Marc Fischlin. Universally composable commitments. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 19–40. Springer, Berlin, Heidelberg, August 2001.

CFR25. Matteo Campanelli, Antonio Faonio, and Luigi Russo. SNARKs for virtual machines are non-malleable. In Serge Fehr and Pierre-Alain Fouque, editors, *EUROCRYPT 2025, Part IV*, volume 15604 of *LNCS*, pages 153–183. Springer, Cham, May 2025.

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, Cham, December 2019.

CKS11. Jan Camenisch, Stephan Krenn, and Victor Shoup. A framework for practical universally composable zero-knowledge protocols. In Dong Hoon Lee and Xiaoyun Wang, editors, *ASIACRYPT 2011*, volume 7073 of *LNCS*, pages 449–467. Springer, Berlin, Heidelberg, December 2011.

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.

CLTZ24. Michele Ciampi, Xiangyu Liu, Ioannis Tzannetos, and Vassilis Zikas. Universal adaptor signatures from blackbox multi-party computation. Cryptology ePrint Archive, Paper 2024/1773, 2024.

CS97. Jan Camenisch and Markus Stadler. Proof systems for general statements about discrete logarithms. Technical Report 260, Department of Computer Science, ETH Zurich, March 1997. Available at ftp://ftp.inf.ethz.ch/pub/publications/tech-reports/.

DeF25. DeFiLlama: Total value locked (TVL). https://defillama.com, 2025. Accessed: 2025-05-13.

DMP88. Alfredo De Santis, Silvio Micali, and Giuseppe Persiano. Non-interactive zero-knowledge proof systems. In Carl Pomerance, editor, *CRYPTO'87*, volume 293 of *LNCS*, pages 52–72. Springer, Berlin, Heidelberg, August 1988.

DOY22. Wei Dai, Tatsuaki Okamoto, and Go Yamamoto. Stronger security and generic constructions for adaptor signatures. In Takanori Isobe and Santanu Sarkar, editors, *INDOCRYPT 2022*, volume 13774 of *LNCS*, pages 52–77. Springer, Cham, December 2022.

DW15. Christian Decker and Roger Wattenhofer. A fast and scalable payment network with bitcoin duplex micropayment channels. In Andrzej Pelc and Alexander A. Schwarzmann, editors, *Stabilization, Safety, and Security of Distributed Systems*, pages 3–18, Cham, 2015. Springer International Publishing.

EEE20. Muhammed F. Esgin, Oguzhan Ersoy, and Zekeriya Erkin. Post-quantum adaptor signatures and payment channel networks. In Liqun Chen, Ninghui Li, Kaitai Liang, and Steve A. Schneider, editors, *ESORICS 2020, Part II*, volume 12309 of *LNCS*, pages 378–397. Springer, Cham, September 2020.

EFH+21. Andreas Erwig, Sebastian Faust, Kristina Hostáková, Monosij Maitra, and Siavash Riahi. Two-party adaptor signatures from identification schemes. In Juan Garay, editor, *PKC 2021, Part I*, volume 12710 of *LNCS*, pages 451–480. Springer, Cham, May 2021.

EFHR20. Lisa Eckey, Sebastian Faust, Kristina Hostáková, and Stefanie Roos. Splitting payments locally while routing interdimensionally. Cryptology ePrint Archive, Report 2020/555, 2020.

Eth25. Ethereum smart contracts stats – Etherscan. https://etherscan.io/contractsVerified, 2025. Accessed: 2025-05-13.

FHH14. Eduarda S. V. Freire, Julia Hesse, and Dennis Hofheinz. Universally composable non-interactive key exchange. In Michel Abdalla and Roberto De Prisco, editors, *SCN 14*, volume 8642 of *LNCS*, pages 1–20. Springer, Cham, September 2014.

Fis05. Marc Fischlin. Communication-efficient non-interactive proofs of knowledge with online extractors. In Victor Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 152–168. Springer, Berlin, Heidelberg, August 2005.

GMM+22. Noemi Glaeser, Matteo Maffei, Giulio Malavolta, Pedro Moreno-Sanchez, Erkan Tairi, and Sri Aravinda Krishnan Thyagarajan. Foundations of coin mixing services. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022*, pages 1259–1273. ACM Press, November 2022.

GMY06. Juan A. Garay, Philip D. MacKenzie, and Ke Yang. Strengthening zero-knowledge protocols using signatures. *Journal of Cryptology*, 19(2):169–209, April 2006.

GSST24. Paul Gerhart, Dominique Schröder, Pratik Soni, and Sri Aravinda Krishnan Thyagarajan. Foundations of adaptor signatures. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part II*, volume 14652 of *LNCS*, pages 161–189. Springer, Cham, May 2024.

HLTW24. Lucjan Hanzlik, Julian Loss, Sri Aravinda Krishnan Thyagarajan, and Benedikt Wagner. Sweep-UC: Swapping coins privately. In *2024 IEEE Symposium on Security and Privacy*, pages 3822–3839. IEEE Computer Society Press, May 2024.

HM04. Dennis Hofheinz and Jörn Müller-Quade. Universally composable commitments using random oracles. In Moni Naor, editor, *TCC 2004*, volume 2951 of *LNCS*, pages 58–76. Springer, Berlin, Heidelberg, February 2004.

HPDB25. Hossein Hafezi, Aditi Partap, Sourav Das, and Joseph Bonneau. Fair signature exchange. Cryptology ePrint Archive, Report 2025/059, 2025.

JCCS19. Dennis Jackson, Cas Cremers, Katriel Cohn-Gordon, and Ralf Sasse. Seems legit: Automated analysis of subtle attacks on protocols that use signatures. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 2165–2180. ACM Press, November 2019.

KOT24. Kaisei Kajita, Go Ohtake, and Tsuyoshi Takagi. Consecutive adaptor signature scheme: From two-party to n-party settings. Cryptology ePrint Archive, Paper 2024/241, 2024. https://eprint.iacr.org/2024/241.

KR17. Ralf Küsters and Daniel Rausch. A framework for universally composable diffie-hellman key exchange. In *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*, pages 881–900. IEEE Computer Society, 2017.

KRDO17. Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 357–388. Springer, Cham, August 2017.

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.

KT09. Ralf Küsters and Max Tuengerthal. Universally composable symmetric encryption. In John C Mitchell, editor, *CSF 2009 Computer Security Foundations Symposium*, pages 293–307. IEEE Computer Society Press, 2009.

KTR20a. Ralf Küsters, Max Tuengerthal, and Daniel Rausch. Joint state composition theorems for public-key encryption and digital signature functionalities with local computation. *Journal of Cryptology*, 33(4):1585–1658, October 2020.

KTR20b. Ralf Küsters, Max Tuengerthal, and Daniel Rausch. The IITM Model: a Simple and Expressive Model for Universal Composability. *Journal of Cryptology*, 33(4):1461–1584, 2020.

MBB+19. Andrew Miller, Iddo Bentov, Surya Bakshi, Ranjit Kumaresan, and Patrick McCorry. Sprites and state channels: Payment networks that go faster than lightning. In Ian Goldberg and Tyler Moore, editors, *FC 2019*, volume 11598 of *LNCS*, pages 508–526. Springer, Cham, February 2019.

MMS+19. Giulio Malavolta, Pedro Moreno-Sanchez, Clara Schneidewind, Aniket Kate, and Matteo Maffei. Anonymous multi-hop locks for blockchain scalability and interoperability. In *NDSS 2019*. The Internet Society, February 2019.

MTV+23. Varun Madathil, Sri Aravinda Krishnan Thyagarajan, Dimitrios Vasilopoulos, Lloyd Fournier, Giulio Malavolta, and Pedro Moreno-Sanchez. Cryptographic oracle-based conditional payments. In *NDSS 2023*. The Internet Society, February 2023.

Poe17. Andrew Poelstra. Scriptless scripts. *Presentation Slides*, 2017.

QPM+23. Xianrui Qin, Shimin Pan, Arash Mirzaei, Zhimei Sui, Oguzhan Ersoy, Amin Sakzad, Muhammed F. Esgin, Joseph K. Liu, Jiangshan Yu, and Tsz Hon Yuen. BlindHub: Bitcoin-compatible privacy-preserving payment channel hubs supporting variable amounts. In *2023 IEEE Symposium on Security and Privacy*, pages 2462–2480. IEEE Computer Society Press, May 2023.

RHK25. Daniel Rausch, Nicolas Huber, and Ralf Küsters. Verifiable E-voting with a trustless bulletin board. In *CSF 2025 Computer Security Foundations Symposium*, pages 457–472. IEEE Computer Society Press, June 2025.

RK24. Farzin Renan and Péter Kutas. SQIAsignHD: SQIsignHD adaptor signature. Cryptology ePrint Archive, Report 2024/561, 2024.

RS09. Markus Rückert and Dominique Schröder. Security of verifiably encrypted signatures and a construction without random oracles. In Hovav Shacham and Brent Waters, editors, *PAIRING 2009*, volume 5671 of *LNCS*, pages 17–34. Springer, Berlin, Heidelberg, August 2009.

Sah99. Amit Sahai. Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In *40th FOCS*, pages 543–553. IEEE Computer Society Press, October 1999.

SHB+22. Benedikt Severin, Marc Hesenius, Florian Blum, Michael Hettmer, and Volker Gruhn. Smart money wasting: Analyzing gas cost drivers of ethereum smart contracts. In *2022 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 293–304, 2022.

SPMS02. Jacques Stern, David Pointcheval, John Malone-Lee, and Nigel P. Smart. Flaws in applying proof methodologies to signature schemes. In Moti Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, pages 93–110. Springer, Berlin, Heidelberg, August 2002.

TMM20. Erkan Tairi, Pedro Moreno-Sanchez, and Matteo Maffei. Post-quantum adaptor signature for privacy-preserving off-chain payments. Cryptology ePrint Archive, Report 2020/1345, 2020.

TMM22. Sri Aravinda Krishnan Thyagarajan, Giulio Malavolta, and Pedro Moreno-Sanchez. Universal atomic swaps: Secure exchange of coins across all blockchains. In *2022 IEEE Symposium on Security and Privacy*, pages 1299–1316. IEEE Computer Society Press, May 2022.

TMS23. Erkan Tairi, Pedro Moreno-Sanchez, and Clara Schneidewind. LedgerLocks: A security framework for blockchain protocols based on adaptor signatures. In Weizhi Meng, Christian Damsgaard Jensen, Cas Cremers, and Engin Kirda, editors, *ACM CCS 2023*, pages 859–873. ACM Press, November 2023.

VST24. Nikhil Vanjani, Pratik Soni, and Sri Aravinda Krishnan Thyagarajan. Functional adaptor signatures: Beyond all-or-nothing blockchain-based payments. In Bo Luo, Xiaojing Liao, Jun Xu, Engin Kirda, and David Lie, editors, *ACM CCS 2024*, pages 1493–1507. ACM Press, October 2024.

WNR20. Pieter Wuille, Jonas Nick, and Tim Ruffing. Schnorr signatures for secp256k1. Bitcoin Improvement Proposal 340, 2020. https://github.com/bitcoin/bips/blob/master/bip-0340.mediawiki.

A Preliminaries

A.1 Hard Relations

A relation $\mathcal{R}$ is a mapping defined as $\mathcal{R} : \mathcal{D}_{\mathsf{S}} \times \mathcal{D}_{\mathsf{W}} \to \{0,1\}$ where $\mathcal{D}_{\mathsf{S}}$ is the space of statements and $\mathcal{D}_{\mathsf{W}}$ is the space of witnesses. Let $Y \in \mathcal{D}_{\mathsf{S}}$ be a statement and $y \in \mathcal{D}_{\mathsf{W}}$ be a witness. $\mathcal{R}$ maps (S, w) to 1 if and only if w is a witness for the statement S. The relation is hard if, with only the statement S given, it is computationally infeasible to compute a witness w such that the relation is satisfied. For practical reasons, verifying the validity of a witness/statement pair and sampling instances (w, s) of the relation should be computationally easy.

A.2 Digital Signatures

A digital signature scheme $\Sigma = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vrfy})$ consists of three efficient algorithms. The key generation algorithm $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KGen}(1^\lambda)$ outputs a secret/public key pair. The signing algorithm $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, x)$ takes a secret key and a message and produces a signature. The verification algorithm $b \leftarrow \mathsf{Vrfy}(\mathsf{pk}, x, \sigma)$ checks whether σ is a valid signature on x under the public key pk.

Definition 3 (EUF-CMA Security). *A signature scheme Σ is* existentially unforgeable under chosen message attacks (EUF-CMA) *if for every* PPT *adversary $\mathcal{A}$, there exists a negligible function* negl *such that for each $\lambda \in \mathbb{N}$*

$$\Pr\left[\mathsf{SigForge}_{\mathcal{A},\Sigma}(\lambda) = 1\right] \leq \mathsf{negl}(\lambda),$$

where the experiment is defined in Figure 6 and the randomness is taken over the random choices of all probabilistic algorithms.

$\mathsf{SigForge}_{\mathcal{A},\Sigma}(\lambda)$	$\mathcal{O}_{\mathsf{Sign}}(x)$
$1:\ \mathcal{Q} := \emptyset$	$1:\ \sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, x)$
$2:\ (\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KGen}(1^\lambda)$	$2:\ \mathcal{Q} = \mathcal{Q} \cup x$
$3:\ (x^*, \sigma^*) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{Sign}}}(\mathsf{pk})$	$3:\ \textbf{return}\ \sigma$
$4:\ \textbf{return}\ (x^* \notin \mathcal{Q} \wedge \mathsf{Vrfy}(\mathsf{pk}, x^*, \sigma^*))$	

Fig. 6. Existential unforgeability experiment for digital signatures.

Definition 4 (Correctness). *A signature scheme Σ satisfies* correctness *if for all $\lambda \in \mathbb{N}$ and all messages $x \in \{0,1\}^*$,*

$$\Pr\left[\mathsf{Vrfy}(\mathsf{pk}, x, \sigma) = 1 \;\middle|\; \begin{array}{l}(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KGen}(1^\lambda),\\ \sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, x) \wedge \\ \sigma \neq \bot\end{array}\right] = 1.$$

A.3 Non-Interactive Zero-Knowledge Proofs

A non-interactive zero-knowledge (NIZK) proof system [DMP88,Fis05] allows a prover to convince a verifier of the truth of a statement without revealing any additional information and without requiring interaction. In this work, we adopt a definition in the *random oracle model* with a *transparent setup*, following [RHK25,BPW12].

Definition 5 (NIZK Proof System in the Random Oracle Model). *Let $\mathcal{R} \subseteq \{0,1\}^* \times \{0,1\}^*$ be an NP relation with associated language $\mathcal{L}_\mathcal{R} := \{x \mid \exists y \text{ such that } \mathcal{R}(x,y) = 1\}$. A* non-interactive zero-knowledge proof system $\mathsf{NIZK} = (\mathsf{P}, \mathsf{V})$ *for $\mathcal{R}$ consists of two probabilistic polynomial-time algorithms with access to a random oracle $\mathcal{RO}$:*
$\pi \leftarrow \mathsf{P}(x,y)$: A prover that, given a statement $x \in \mathcal{L}_\mathcal{R}$ and a corresponding witness y, outputs a proof $\pi \in \{0,1\}^*$. The prover may query the random oracle $\mathcal{RO}$.
$b \leftarrow \mathsf{V}(x,\pi)$: A verifier that, given a statement x from an efficiently decidable superset $\Lambda \supseteq \mathcal{L}_\mathcal{R}$ and a candidate proof π, returns a bit $b \in \{0,1\}$. The verifier may also query the random oracle $\mathcal{RO}$ and accepts only if it is convinced that $x \in \mathcal{L}_\mathcal{R}$.

Definition 6 (Zero-Knowledge Simulator [Sah99]). *A simulator* Sim *for a* NIZK*proof system is a PPT algorithm that responds to queries to a random oracle and produces simulated proofs for statements $x \in \Lambda$, where $\Lambda \supseteq \mathcal{L}_\mathcal{R}$ is an efficiently decidable superset of valid statements. The simulator may generate accepting proofs even for statements outside the language, i.e., $x \notin \mathcal{L}_\mathcal{R}$. It supports the following two types of queries:*

- *$\mathcal{RO}(y)$: The simulator maintains a list of oracle query/response pairs (y,r). On input y, it returns the same r as before if the query has already been made. Otherwise, it samples a fresh random r from the oracle range, records (y,r), and returns r.*
- *$\mathsf{Simulate}(x)$: For any $x \in \Lambda$, the simulator returns a proof π such that $\mathsf{V}^{\mathsf{Sim}}(x,\pi) = 1$, where the verifier uses the simulator's programmed oracle. To ensure validity, the simulator may program the oracle by adding appropriate entries to its internal query/response list.*

We require the following security properties from NIZK proof systems:

- **(Perfect) Completeness:** For all $x \in \mathcal{L}_\mathcal{R}$ and corresponding witnesses y, a valid proof is accepted with probability 1:
$$\Pr[\mathsf{V}(x,\pi) = 1 \mid \pi \leftarrow \mathsf{P}(x,y)] = 1.$$
- **Soundness:** For any $x \notin \mathcal{L}_\mathcal{R}$, it is computationally infeasible to produce an accepting proof:
$$\Pr[\mathsf{V}(wx,\pi) = 1 \mid \pi \leftarrow \mathcal{A}(x)] \leq \mathsf{negl}(\lambda).$$
- **Zero-Knowledge:** There exists a simulator Sim (with programming access to $\mathcal{RO}$) such that simulated proofs are computationally indistinguishable from real ones:
$$\left|\Pr\left[\mathcal{A}^{\mathsf{Sim}(x)} = 1\right] - \Pr\left[\mathcal{A}^{\mathsf{V}(x,\pi)} = 1\right]\right| \leq \mathsf{negl}(\lambda),$$
where $\pi \leftarrow \mathsf{P}(x,y)$ and $\mathsf{Sim}(x)$ denotes a simulated proof generated without y.
- **Simulation-Sound Extractability [BPW12,CFR25]:** If NIZK is zero-knowledge with respect to a simulator Sim, and there exists a PPT extractor Ext such that for any efficient adversary $\mathcal{A}$, the following game is won by Ext with overwhelming probability, then we say that NIZK satisfies *simulation-sound extractability.*
  1. *Initial run:* The game samples a random tape ω and runs $\mathcal{A}(\omega)$ against the simulator Sim, which answers random oracle queries and simulation requests. The adversary outputs a list of statement/proof pairs $\{(x_i,\pi_i)\}_i$. If any (x_i,π_i) fails verification or π_i was generated by a previous $\mathsf{Simulate}(x_i)$ query, the extractor wins trivially.
  2. *Extraction phase:* The extractor Ext receives the transcript of the initial run and the adversary's output $\{(x_i,\pi_i)\}_i$. It may issue Invoke queries, which re-run $\mathcal{A}$ with the same randomness ω. All queries made during these invocations are forwarded to Ext.
  3. *Winning:* The extractor succeeds if it outputs witnesses $\{w_i\}_i$ such that for all i, $(x_i,w_i) \in \mathcal{R}$.

A.4 Schnorr-Style Proofs of Knowledge

We instantiate our proofs using sigma protocols for the discrete logarithm (DLog) relation, following Camenisch and Stadler [CS97], who proved completeness, honest-verifier zero-knowledge, and special soundness for this class. When combined with the Fiat–Shamir transform, these interactive protocols become non-interactive, retain their zero-knowledge property [CS97], and are extractable using the forking lemma [BN06,BPW12]. For straight-line extractability, the Fischlin transform [Fis05] can be applied instead of the Fiat-Shamir transform.

B Adaptor Signature Preliminaries

B.1 Schnorr Adaptor Signature Construction

Construction 2 (Schnorr Adaptor Signatures) *Let $\Sigma = (\mathsf{Sign}, \mathsf{Vrfy})$ be the Schnorr signature scheme defined over a group $\mathbb{G}$ of prime order p with generator g. Let $\mathcal{R} = \{(Y, y) \mid Y = g^y\}$ be the discrete logarithm relation. We define the adaptor signature scheme $\mathsf{AS} = (\mathsf{pSign}, \mathsf{Adapt}, \mathsf{pVrfy}, \mathsf{Extract})$ in Figure 7, which extends Schnorr signatures to embed a witness y for a statement $Y \in \mathbb{G}$.*

$\mathsf{KGen}(\lambda)$

1 : $\mathsf{sk} \leftarrow\!\!\$\ \mathbb{Z}_p$
2 : $\mathsf{pk} := g^{\mathsf{sk}}$
3 : **return** $(\mathsf{sk}, \mathsf{pk})$

$\mathsf{Sign}(\mathsf{sk}, m)$

1 : $r \leftarrow\!\!\$\ \mathbb{Z}_p;\ R := g^r$
2 : $c := \mathsf{H}_{\mathsf{Sign}}(\mathsf{pk}, m, R)$
3 : $s := r + c \cdot \mathsf{sk}$
4 : **return** (R, s)

$\mathsf{Vrfy}(\mathsf{pk}, m, \sigma)$

1 : $(R, s) := \sigma$
2 : $c := \mathsf{H}_{\mathsf{Sign}}(\mathsf{pk}, m, R)$
3 : **return** $[g^s = R \cdot \mathsf{pk}^c]$

$\mathsf{Extract}(\widetilde{\sigma}, \sigma)$

1 : $(R \cdot Y, s) := \widetilde{\sigma}$
2 : $(R \cdot Y, s') := \sigma$
3 : **return** $s' - s$

$\mathsf{pSign}(\mathsf{sk}, m, Y)$

1 : $r \leftarrow\!\!\$\ \mathbb{Z}_p;\ R := g^r$
2 : $c := \mathsf{H}_{\mathsf{Sign}}(\mathsf{pk}, m, R \cdot Y)$
3 : $s := r + c \cdot \mathsf{sk}$
4 : **return** $(R \cdot Y, s)$

$\mathsf{pVrfy}(\mathsf{pk}, m, Y, \widetilde{\sigma})$

1 : $(R \cdot Y, s) := \widetilde{\sigma}$
2 : $c := \mathsf{H}_{\mathsf{Sign}}(\mathsf{pk}, m, R \cdot Y)$
3 : **return** $[g^s \cdot Y = R \cdot Y \cdot \mathsf{pk}^c]$

$\mathsf{Adapt}(\widetilde{\sigma}, y)$

1 : $(R \cdot Y, s) := \widetilde{\sigma}$
2 : **return** $(R \cdot Y, s + y)$

Fig. 7. Schnorr Adaptor Signature Construction [AEE+21].

B.2 Security of Adaptor Signatures

Correctness. The correctness of adaptor signatures is captured by two properties: *pre-signature correctness* and *pre-signature adaptability*. Previous work defined pre-signature correctness as a combined property encompassing three guarantees. First, classical correctness: any honestly generated pre-signature should successfully pass pre-verification. Second, completeness of adaptation: such a pre-signature, when combined with a valid witness, should adapt to a valid full signature. Third, completeness of extraction: given a valid pre-signature and a corresponding adapted signature, the extract algorithm should return a valid witness for the associated statement. In our treatment, we explicitly split correctness into two distinct properties.

As first property, we assume pre-sign correctness. An adaptor signature is correct if, for an honestly generated keypair, pre-signing any message–statement pair results in a valid pre-signature w.r.t. this message–statement pair. This should hold for all bitstrings of statements, as long as such a statement is a valid input for the pre-signing algorithm.

Definition 7 (Pre-Sign Correctness). *An adaptor signature scheme* AS *satisfies* pre-sign correctness *if for all* $\lambda \in \mathbb{N}$ *and all messages* $x \in \{0,1\}^*$ *and all* $Y \in \{0,1\}^*$*,*

$$\Pr\left[\mathsf{pVrfy}(\mathsf{pk}, x, Y, \widetilde{\sigma}) = 1 \;\middle|\; \begin{array}{l}(\mathsf{sk},\mathsf{pk}) \leftarrow \mathsf{KGen}(1^\lambda),\\ \widetilde{\sigma} \leftarrow \mathsf{pSign}(\mathsf{sk}, x, Y) \wedge \\ \widetilde{\sigma} \neq \bot\end{array}\right] = 1.$$

As a second property, we assume extract correctness. Extract correctness enforces that when a pre-signature w.r.t. some statement Y is adapted using a valid witness, the pre-signature–adapted-pre-signature pair allows extracting a valid witness for Y.

Definition 8 (Extract Correctness). *An adaptor signature scheme* AS *satisfies* extract correctness *if for all messages* $x \in \{0,1\}^*$*, all* $(Y, y) \in \mathcal{R}$*, and all public keys* pk *and pre-signatures* $\widetilde{\sigma} \in \{0,1\}^*$*,*

$$\Pr\left[(Y, y') \in \mathcal{R} \;\middle|\; \begin{array}{l}\sigma := \mathsf{Adapt}(\mathsf{pk}, \widetilde{\sigma}, y, Y) \wedge \\ y' := \mathsf{Extract}(\mathsf{pk}, \widetilde{\sigma}, \sigma, Y)\end{array}\right] = 1.$$

While our two-phase correctness notion is technically stronger than the established one as extract correctness is required for all signatures that verify rather than only those returned by the signing algorithm, it better reflects the intuitive design goal of a correct adaptor signature scheme and is captured by most adaptor signature schemes where the extract algorithm is essentially the inverse function of the adapt algorithm. Compared to the correctness definition of [GSST24], we additionally relax the original restriction to fixed-length messages and now require this property for arbitrary messages (in both our correctness definitions). This change does not actually impose stronger requirements but is rather a generalization: any scheme that is correct for messages of a specific fixed length can be interpreted as a scheme that is correct on $\{0,1\}^*$ by having (pre-)sign and (pre-)verify algorithms reject inputs when messages do not adhere to the original fixed length. The main reason for this modification is that it aligns more closely with UC security, which requires properties to hold true for all possible inputs that the environment might provide. We show in Section 5.1 that our Schnorr adaptor signature scheme satisfies this two-part definition.

Pre-signature adaptability. Pre-signature adaptability guarantees that if a pre-signature is valid w.r.t. an honestly generated statement, then adapting such a pre-signature with a valid witness for the statement yields a valid signature. We also relax the restriction for fixed-length messages, due to the same reasons as for correctness.

Definition 9 (Pre-signature adaptability). *An adaptor signature scheme* AS *satisfies* pre-signature adaptability *if for all* $\lambda \in \mathbb{N}$*, all messages* $x \in \{0,1\}^*$*, all* $(Y, y) \in \mathcal{R}$*, and all public keys* pk *and pre-signatures* $\widetilde{\sigma} \in \{0,1\}^*$*, it holds that if* $\mathsf{pVrfy}(\mathsf{pk}, x, Y, \widetilde{\sigma}) = 1$*, then* $\mathsf{Vrfy}(\mathsf{pk}, x, \mathsf{Adapt}(\mathsf{pk}, \widetilde{\sigma}, y, Y)) = 1$*.*

Extractability. Extractability extends the notion of existential unforgeability from standard signature schemes to the adaptor signature setting. As in unforgeability, the adversary wins if it can produce a valid signature on a message that was not previously signed by the signing oracle. However, in the adaptor setting, the adversary is also granted access to a pre-signing oracle for any message–statement pair, as well as an oracle that samples fresh statements from the hard relation. The winning condition is strengthened: the adversary must produce a signature on a fresh message that not only verifies, but also cannot be used—together with any pre-signature obtained from the pre-signing oracle—to extract a valid witness for the associated statement.

Definition 10 (Extractability). *An adaptor signature scheme* AS *satisfies* extractability *if for every* PPT *adversary* $\mathcal{A}$*, there exists a negligible function* negl *such that for every* $\lambda \in \mathbb{N}$*,*

$$\Pr\left[\mathsf{Ext}_{\mathcal{A},\mathsf{AS}}(\lambda) = 1\right] \leq \mathsf{negl}(\lambda)\ ,$$

where the experiment $\mathsf{Ext}_{\mathcal{A},\mathsf{AS}}$ *is defined in Figure 8, and the probability is taken over the randomness of all probabilistic algorithms.*

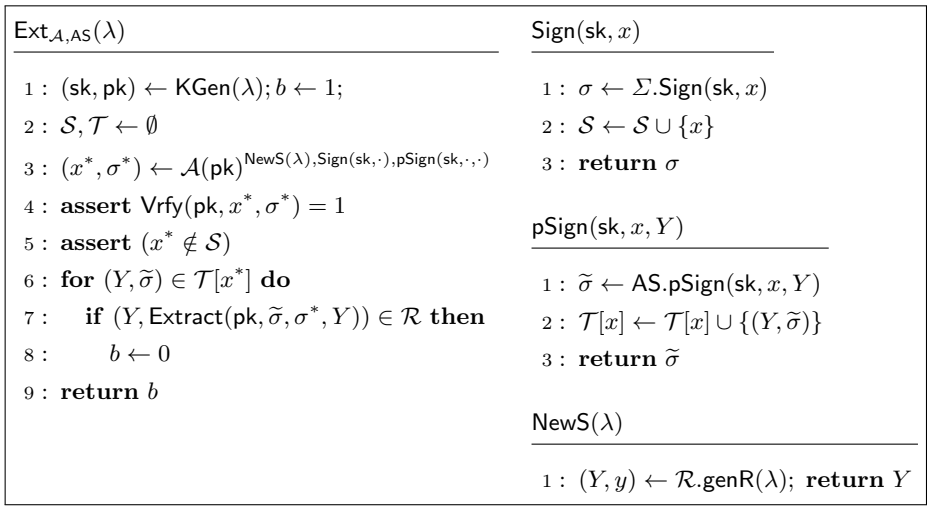

Fig. 8. The extractability experiment $\mathsf{Ext}_{\mathcal{A},\mathsf{AS}}(\lambda)$.

Unique extractability. Unique extractability guarantees that any valid pre-signature acts as a commitment to both a *single* valid signature and a *single* witness. That is, no efficient adversary can produce a pre-signature $\widetilde{\sigma}$ for a message x and statement Y such that there exist two distinct signatures on x that both allow extraction of valid witnesses (with respect to Y) from the same pre-signature. More formally:

Definition 11 (Unique Extractability). *An adaptor signature scheme* AS *satisfies* unique extractability *if for every* PPT *adversary* $\mathcal{A}$*, there exists a negligible function* negl *such that for every* $\lambda \in \mathbb{N}$*,*

$$\Pr\left[\mathsf{UniqueExtractability}_{\mathcal{A},\mathsf{AS}}(\lambda) = 1\right] \leq \mathsf{negl}(\lambda) \ ,$$

where the experiment $\mathsf{UniqueExtractability}_{\mathcal{A},\mathsf{AS}}$ *is defined in Figure 9, and the probability is taken over the randomness of all probabilistic algorithms.*

As discussed in the technical outline, we omit the notions of unlinkability and pre-verify soundness from this work. For completeness, we provide the former correctness notion [GSST24].

Definition 12 (Pre-signature correctness [GSST24]). *An adaptor signature scheme* AS *satisfies* pre-signature correctness *if for all* $\lambda \in \mathbb{N}$ *and all messages* $x \in \{0,1\}^{\ell_m}$*,*

$$\Pr\left[\begin{array}{l|l} \mathsf{pVrfy}(\mathsf{pk}, x, Y, \widetilde{\sigma}) = 1 \wedge & (\mathsf{sk},\mathsf{pk}) \leftarrow \mathsf{KGen}(1^\lambda), \\ \mathsf{Vrfy}(\mathsf{pk}, x, \sigma) = 1 \wedge & (Y, y) \leftarrow \mathsf{genR}(1^\lambda), \\ (Y, y') \in \mathcal{R} & \widetilde{\sigma} \leftarrow \mathsf{pSign}(\mathsf{sk}, x, Y), \\ & \sigma := \mathsf{Adapt}(\mathsf{pk}, \widetilde{\sigma}, y), \\ & y' := \mathsf{Extract}(\mathsf{pk}, \widetilde{\sigma}, \sigma, Y) \end{array}\right] = 1.$$

C Realization Proof of $\mathcal{F}_{\mathbf{asig}}$

Proof (Proof of Theorem 2). To prove Theorem 2, we have to show the existence of a simulator $\mathcal{S}$ such that the real world protocol that just runs $\mathsf{AS}_{\Sigma,\mathcal{R}}$ (in the presence of the dummy adversary that simply forwards network messages) is indistinguishable from the ideal functionality $\mathcal{F}_{\text{asig}}$ running with the simulator $\mathcal{S}$ for all ppt environments. As is usual for UC security, we only have to show this property for a single session/copy of the protocol (identified by an arbitrary but fixed challenge session ID *sid*) and hence only for one key pair that, however, may be used by an arbitrary number of parties. A general composition theorem then implies

$\mathsf{UniqueExtractability}_{\mathcal{A},\mathsf{AS}}(\lambda)$

1 : $(\mathsf{sk},\mathsf{pk}) \leftarrow \mathsf{KGen}(\lambda)$
2 : $(x, Y, \widetilde{\sigma}, \sigma_1, \sigma_2) \leftarrow \mathcal{A}^{\mathsf{Sign}(\mathsf{sk},\cdot),\mathsf{pSign}(\mathsf{sk},\cdot,\cdot)}(\mathsf{pk})$
3 : **assert** $(\sigma_1 \neq \sigma_2)$
4 : **assert** $\mathsf{Vrfy}(\mathsf{pk}, x, \sigma_1) = 1$
5 : **assert** $\mathsf{Vrfy}(\mathsf{pk}, x, \sigma_2) = 1$
6 : **assert** $\mathsf{pVrfy}(\mathsf{pk}, x, Y, \widetilde{\sigma}) = 1$
7 : $y_1 \leftarrow \mathsf{Extract}(\mathsf{pk}, \widetilde{\sigma}, \sigma_1, Y)$
8 : $y_2 \leftarrow \mathsf{Extract}(\mathsf{pk}, \widetilde{\sigma}, \sigma_2, Y)$
9 : **return** $(Y, y_1) \in \mathcal{R} \wedge (Y, y_2) \in \mathcal{R}$

$\mathsf{Sign}(\mathsf{sk}, x)$

1 : $\sigma \leftarrow \Sigma.\mathsf{Sign}(\mathsf{sk}, x)$
2 : **return** σ

$\mathsf{pSign}(\mathsf{sk}, x, Y)$

1 : $\widetilde{\sigma} \leftarrow \mathsf{AS}.\mathsf{pSign}(\mathsf{sk}, x, Y)$
2 : **return** $\widetilde{\sigma}$

Fig. 9. The unique extractability experiment $\mathsf{UniqueExtractability}_{\mathcal{A},\mathsf{AS}}(\lambda)$.

that UC security is retained even when an environment uses polynomially many different copies/sessions and hence key pairs (see, e.g., Theorem 3 in [KTR20b], which shows this as an explicit and distinct property, or the definition of the UC security experiment and Theorem 22 given in [Can20], which captures this implicitly along with other properties).

We show this theorem for $\mathcal{S}$ being the dummy adversary, i.e., the adversary in real and ideal world being identical. We thus only have to show that $\mathsf{AS}_{\Sigma,\mathcal{R}}$ and $\mathcal{F}_{\text{asig}}$ are indistinguishable for all environments without having to modify the adversary as part of the proof. The proof itself is structured as a sequence of games that start from the real protocol $\mathsf{AS}_{\Sigma,\mathcal{R}}$ and iteratively modify it until we reach $\mathcal{F}_{\text{asig}}$. We reduce each game hop to one or more of the game-based properties required by Theorem 2 to show that the distinguishing probability for a ppt environment changes only by a negligible probability. Since the number of game hops is constant, this immediately gives the claim, i.e., that the real protocol is indistinguishable from the ideal one.

As mentioned, our starting point is the real protocol that just runs $\mathsf{AS}_{\Sigma,\mathcal{R}}$ as is, which we formalize via the protocol $\mathcal{F}^0_{\text{asig}}$ given in Figure 10. This is basically the same as Figure 2 that we used as a starting point for our technical overview given in Section 4. The protocol $\mathcal{F}^0_{\text{asig}}$ is parameterized by a polytime decidable relation $\mathcal{R}$, ppt algorithms $\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{pSign}$, and dpt algorithms $\mathsf{Vrfy}, \mathsf{pVrfy}, \mathsf{Adapt}, \mathsf{Extract}$. When an oracle of the protocol is called, the underlying corresponding algorithm (indicated by the same name) is executed and the result is returned. Since only the owner of the key pair knows the secret key, only he can call the Sign and pSign oracles.

Game $\mathcal{G}_1$. In the first game, we start by modifying our real protocol to obtain a first intermediate ideal functionality that enforces correctness properties in the verification, pre-verification, and extract oracles. To this extent, our functionality deploys bookkeeping. This means, when a signature or pre-signature is successfully created, the functionality stores the corresponding (pre-)signatures into lists $\mathsf{H}_{\mathsf{sign}}$ and $\mathsf{H}_{\mathsf{pSign}}$. When a signature or pre-signature is verified, the functionality accepts all signatures from $\mathsf{H}_{\mathsf{sign}}$ and all pre-signatures from $\mathsf{H}_{\mathsf{pSign}}$. In addition, the functionality stores the results of successfully pre-verifying pre-signatures in $\mathsf{H}_{\mathsf{pVerify}}$. As an additional list, we introduce the list $\mathsf{H}_{\mathsf{adapt}}$. In $\mathsf{H}_{\mathsf{adapt}}$, we store the inputs and the output of the adapt oracle, i.e., the pairs $(\widetilde{\sigma}, Y, y, \sigma)$, when the pre-signature–statement pair is stored in $\mathsf{H}_{\mathsf{pVerify}}$, the witness y is a valid witness for Y, and the signature σ is the output of calling the adapt algorithm on $(\widetilde{\sigma}, Y, y)$. Looking ahead, this list will allow us to enforce extract correctness, as well as pre-signature adaptability. Besides this, the functionality $\mathcal{F}^1_{\text{asig}}$ has the same outputs as $\mathcal{F}^0_{\text{asig}}$. We depict $\mathcal{F}^1_{\text{asig}}$ in Figure 11.

The functionality is parameterized by a polytime decidable relation $\mathcal{R}$, ppt algorithms $\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{pSign}$ and dpt algorithms $\mathsf{Vrfy}, \mathsf{pVrfy}, \mathsf{Adapt}, \mathsf{Extract}$. Upon first activation, initialize this instance by running and storing $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$.

On $\mathtt{PubKey}$? from anyone, return $(\mathtt{PubKey}, \mathsf{pk})$.

On $(\mathtt{Sign}, x)$ for party pid in session $sid = (pid, sid')$, do:
$\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, x)$
Return $(\mathtt{Signature}, \sigma)$.

On $(\mathtt{pSign}, x, Y)$ for party pid in session $sid = (pid, sid')$, do:
$\widetilde{\sigma} \leftarrow \mathsf{pSign}(\mathsf{sk}, x, Y)$
Return $(\mathtt{PreSignature}, \widetilde{\sigma})$

On $(\mathtt{Adapt}, pk, \widetilde{\sigma}, y, Y)$ for party pid, do:
$\sigma \leftarrow \mathsf{Adapt}(pk, \widetilde{\sigma}, y, Y)$
Return $(\mathtt{Adapt}, \sigma)$

On $(\mathtt{Extract}, pk, \widetilde{\sigma}, \sigma, Y)$ for party pid, do:
$y := \mathsf{Extract}(pk, \widetilde{\sigma}, \sigma, Y)$
Return $(\mathtt{Extract}, y)$

On $(\mathtt{pVerify}, pk, x, Y, \widetilde{\sigma})$ for party pid, do:
$b_{\mathsf{pVerify}} := \mathsf{pVrfy}(pk, x, Y, \widetilde{\sigma})$
Return $(\mathtt{VerResult}, b_{\mathsf{pVerify}})$

On $(\mathtt{Verify}, pk, x, \sigma)$ for party pid, do:
$b_{\mathsf{vrfy}} := \mathsf{Vrfy}(pk, x, \sigma)$
Return $(\mathtt{VerResult}, b_{\mathsf{vrfy}})$

As usual, the attacker may corrupt any party pid dynamically and gains full control over that party. In addition, if pid is the owner of the key pair modeled by this instance, i.e., the session ID of this instance is $sid = (pid, sid')$, then the attacker also learns sk; for notational convenience, we set $\mathsf{corr} = \mathsf{true}$ iff this is the case.

Fig. 10. Ideal adaptor signatures functionality $\mathcal{F}_{\text{asig}}^0$.

The functionality is parameterized by a polytime decidable relation $\mathcal{R}$, ppt algorithms $\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{pSign}$ and dpt algorithms $\mathsf{Vrfy}, \mathsf{pVrfy}, \mathsf{Adapt}, \mathsf{Extract}$. Upon first activation, initialize this instance by running and storing $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$.

On $\texttt{PubKey}$? from anyone, return $(\texttt{PubKey}, \mathsf{pk})$.

On $(\texttt{Sign}, x)$ for party pid in session $sid = (pid, sid')$, do:

$\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, x)$
if $\sigma \neq \bot$: **add** σ **to** $\mathsf{H}_{\mathsf{sign}}[x]$

Return $(\texttt{Signature}, \sigma)$.

On $(\texttt{pSign}, x, Y)$ for party pid in session $sid = (pid, sid')$, do:

$\widetilde{\sigma} \leftarrow \mathsf{pSign}(\mathsf{sk}, x, Y)$
if $\widetilde{\sigma} \neq \bot$: **add** $(\widetilde{\sigma}, Y)$ **to** $\mathsf{H}_{\mathsf{pSign}}[x]$

Return $(\texttt{PreSignature}, \widetilde{\sigma})$

On $(\texttt{Adapt}, pk, \widetilde{\sigma}, y, Y)$ for party pid, do:

$\sigma \leftarrow \mathsf{Adapt}(pk, \widetilde{\sigma}, y, Y)$
if $pk = \mathsf{pk} \wedge (Y, y) \in \mathcal{R}$: **add** $(\widetilde{\sigma}, Y, y, \sigma)$ **to** $\mathsf{H}_{\mathsf{adapt}}$

Return $(\texttt{Adapt}, \sigma)$

On $(\texttt{Extract}, pk, \widetilde{\sigma}, \sigma, Y)$ for party pid, do:

$y := \mathsf{Extract}(pk, \widetilde{\sigma}, \sigma, Y)$
if $(Y, y) \notin \mathcal{R} \wedge \exists (\widetilde{\sigma}, Y, y', \sigma) \in \mathsf{H}_{\mathsf{adapt}}$: $y := y'$ { *Extract Correctness*

Return $(\texttt{Extract}, y)$

On $(\texttt{pVerify}, pk, x, Y, \widetilde{\sigma})$ for party pid, do:

$b_{\mathsf{pVerify}} := \mathsf{pVrfy}(pk, x, Y, \widetilde{\sigma})$
if $pk = \mathsf{pk} \wedge (\widetilde{\sigma}, Y) \in \mathsf{H}_{\mathsf{pSign}}[x]$: $b_{\mathsf{pVerify}} := 1$ { *Pre-Sign Correctness*
if $pk = \mathsf{pk} \wedge b_{\mathsf{pVerify}}$: **add** $(\widetilde{\sigma}, Y)$ $\mathsf{H}_{\mathsf{pVerify}}[x]$ **to**

Return $(\texttt{VerResult}, b_{\mathsf{pVerify}})$

On $(\texttt{Verify}, pk, x, \sigma)$ for party pid, do:

$b_{\mathsf{vrfy}} := \mathsf{Vrfy}(pk, x, \sigma)$
if $pk = \mathsf{pk} \wedge \in \mathsf{H}_{\mathsf{sign}}[x]$: $b_{\mathsf{vrfy}} := 1$ { *Correctness*

Return $(\texttt{VerResult}, b_{\mathsf{vrfy}})$

As usual, the attacker may corrupt any party pid dynamically and gains full control over that party. In addition, if pid is the owner of the key pair modeled by this instance, i.e., the session ID of this instance is $sid = (pid, sid')$, then the attacker also learns sk; for notational convenience, we set $\mathsf{corr} = \mathsf{true}$ iff this is the case.

Fig. 11. Ideal adaptor signatures functionality $\mathcal{F}_{\mathrm{asig}}^{1}$.

Claim 1 *If* AS *is pre-sign correct (Definition 7), has correct extracting (Definition 8), and* Σ *is a correct signature scheme (Definition 4), then the gap between* $\mathcal{F}^1_{asig}$ *and* $\mathcal{F}^0_{asig}$ *is negligible.*

Proof. To show Claim 1, we note that $\mathcal{F}^1_{\text{asig}}$ and $\mathcal{F}^0_{\text{asig}}$ differ in three cases. The first case is, if a signature on a message x was obtained via $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, x)$ by an oracle, and $\mathsf{Vrfy}(\mathsf{pk}, x, \sigma) = 0$. As the signature scheme Σ is (perfectly) correct, $\mathsf{Vrfy}(\mathsf{pk}, x, \mathsf{Sign}(\mathsf{sk}, x)) = 0$ cannot happen. The second case is equivalent to the first, but when pre-verifying pre-signatures output by the pre-sign oracle. This second change is again undetectable by the perfect pre-sign correctness of the adaptor signature scheme. In the third case, the functionalities differ if the extract algorithm outputs bot on a pre-signature–signature–statement tuple, but this tuple was added to $\mathsf{H}_{\mathsf{adapt}}$. To be added to $\mathsf{H}_{\mathsf{adapt}}$, the signature is the output of the adapt oracle called on a pre-signature, and a valid statement–witness pair. When the functionality overwrites a witness, the extract algorithm failed to provide a valid witness for the output of adapt. If this case occurs, it would thus contradict extract correctness. Since extract correctness is a perfect property, this case cannot happen. Hence, the gap between $\mathcal{F}^1_{\text{asig}}$ and $\mathcal{F}^0_{\text{asig}}$ is bounded by correctness, pre-sign correctness, and extract correctness. Since these are perfect properties, this gap is 0.

At this point in our series of game hops, whenever an oracle considers a pre-signature or signature valid w.r.t. some message (and some statement), then the actual algorithms output that the (pre-)signature is valid. Hence, all signatures in the set $\mathsf{H}_{\mathsf{sign}}[x]$ will verify w.r.t. pk and x, and all pre-signature–statement pairs $(\widetilde{\sigma}, Y)$ in the sets $\mathsf{H}_{\mathsf{pSign}}[x]$ and $\mathsf{H}_{\mathsf{pVerify}}[x]$ will pre-verify w.r.t. x. This holds since from this point forward, we will not add more elements to these lists.

Game $\mathcal{G}_2$**.** In this game, we enforce pre-signature adaptability. This means that, in this functionality, each signature obtained by adapting a pre-verifying pre-signature is guaranteed to verify. Besides this, the functionality $\mathcal{F}^2_{\text{asig}}$ has the same outputs as $\mathcal{F}^1_{\text{asig}}$. We depict $\mathcal{F}^2_{\text{asig}}$ in Figure 12.

Claim 2 *If* AS*, in addition to all the properties required by previous claims, satisfies pre-signature adaptability Definition 9, then the gap between* $\mathcal{F}^2_{asig}$ *and* $\mathcal{F}^1_{asig}$ *is negligible.*

Proof. The outputs of the functionalities $\mathcal{F}^2_{\text{asig}}$ and $\mathcal{F}^1_{\text{asig}}$ differ if the verification algorithm is called on a signature and a message x, such that the signature should be invalid by the verification algorithm, but the signature is part of the set $\mathsf{H}_{\mathsf{adapt}}$, and there exits a tuple $(\widetilde{\sigma}, Y, y)$, such that the pre-signature $\widetilde{\sigma}$ was pre-verified w.r.t. Y and x. This means that $\mathcal{F}^2_{\text{asig}}$ differs from $\mathcal{F}^1_{\text{asig}}$, if a valid pre-signature w.r.t. some message x and some statement Y yields into an invalid adapted signature when adapted using a valid witness y for Y–effectively violating pre-signature adaptability. As we assume AS to have (perfect) pre-signature adaptability, there is no gap between $\mathcal{F}^2_{\text{asig}}$ and $\mathcal{F}^1_{\text{asig}}$.

Game $\mathcal{G}_3$**.** In this game, we enforce unforgeability for pre-signatures for as long as the signer is uncorrupted. This means that, in this functionality, a pre-signature can only be valid if it was either pre-signed before or if the message on which this pre-signature is pre-verified was signed before. Otherwise, the pre-verification algorithm outputs 0 in $\mathcal{F}^3_{\text{asig}}$. Besides this, the functionality $\mathcal{F}^3_{\text{asig}}$ has the same outputs as $\mathcal{F}^2_{\text{asig}}$. We depict $\mathcal{F}^3_{\text{asig}}$ in Figure 13.

Claim 3 *If* AS*, in addition to all the properties required by previous claims, has unforgeable pre-signatures (Definition 2), then the gap between* $\mathcal{F}^3_{asig}$ *and* $\mathcal{F}^2_{asig}$ *is negligible.*

Proof. The outputs of the functionalities $\mathcal{F}^3_{\text{asig}}$ and $\mathcal{F}^2_{\text{asig}}$ differ only if the signer is uncorrupted, and the pre-verification algorithm is called on a pre-signature and a message x, such that the pre-signature should be valid by the pre-verification algorithm, but the message x was never signed before, and the message-statement pair in question was not pre-signed before. In such cases, $\mathcal{F}^3_{\text{asig}}$ does not qualify the pre-signature to be valid, while $\mathcal{F}^2_{\text{asig}}$ would. We now show that the probability of such cases occurring is negligible in the security parameter. To this end, we provide a reduction that transforms any environment causing this deviation into an adversary that breaks the pre-signature unforgeability of AS. As input, our reduction receives a public key and has access to a signing and a pre-signing oracle that outputs valid (pre-)signatures w.r.t. the

The functionality is parameterized by a polytime decidable relation $\mathcal{R}$, ppt algorithms $\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{pSign}$ and dpt algorithms $\mathsf{Vrfy}, \mathsf{pVrfy}, \mathsf{Adapt}, \mathsf{Extract}$. Upon first activation, initialize this instance by running and storing $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$.

On $\texttt{PubKey}$? from anyone, return $(\texttt{PubKey}, \mathsf{pk})$.

On $(\texttt{Sign}, x)$ for party pid in session $sid = (pid, sid')$, do:
$\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, x)$
if $\sigma \neq \bot$**: add** σ **to** $\mathsf{H}_{\mathsf{sign}}[x]$
Return $(\texttt{Signature}, \sigma)$.

On $(\texttt{pSign}, x, Y)$ for party pid in session $sid = (pid, sid')$, do:
$\widetilde{\sigma} \leftarrow \mathsf{pSign}(\mathsf{sk}, x, Y)$
if $\widetilde{\sigma} \neq \bot$**: add** $(\widetilde{\sigma}, Y)$ **to** $\mathsf{H}_{\mathsf{pSign}}[x]$
Return $(\texttt{PreSignature}, \widetilde{\sigma})$

On $(\texttt{Adapt}, pk, \widetilde{\sigma}, y, Y)$ for party pid, do:
$\sigma \leftarrow \mathsf{Adapt}(pk, \widetilde{\sigma}, y, Y)$
if $pk = \mathsf{pk} \wedge (Y, y) \in \mathcal{R}$**: add** $(\widetilde{\sigma}, Y, y, \sigma)$ **to** $\mathsf{H}_{\mathsf{adapt}}$
Return $(\texttt{Adapt}, \sigma)$

On $(\texttt{Extract}, pk, \widetilde{\sigma}, \sigma, Y)$ for party pid, do:
$y := \mathsf{Extract}(pk, \widetilde{\sigma}, \sigma, Y)$
if $(Y, y) \notin \mathcal{R} \wedge \exists (\widetilde{\sigma}, Y, y', \sigma) \in \mathsf{H}_{\mathsf{adapt}}$**:** $y := y'$ { *Extract Correctness*
Return $(\texttt{Extract}, y)$

On $(\texttt{pVerify}, pk, x, Y, \widetilde{\sigma})$ for party pid, do:
$b_{\mathsf{pVerify}} := \mathsf{pVrfy}(pk, x, Y, \widetilde{\sigma})$
if $pk = \mathsf{pk} \wedge (\widetilde{\sigma}, Y) \in \mathsf{H}_{\mathsf{pSign}}[x]$**:** $b_{\mathsf{pVerify}} := 1$ { *Pre-Sign Correctness*
else if $pk = \mathsf{pk} \wedge \neg\mathsf{corr} \wedge \mathsf{H}_{\mathsf{sign}}[x] = \emptyset$**:** $b_{\mathsf{pVerify}} := 0$ { *Pre-Sig Unforgeability*
if $pk = \mathsf{pk} \wedge b_{\mathsf{pVerify}}$**: add** $(\widetilde{\sigma}, Y)$ $\mathsf{H}_{\mathsf{pVerify}}[x]$ **to**
Return $(\texttt{VerResult}, b_{\mathsf{pVerify}})$

On $(\texttt{Verify}, pk, x, \sigma)$ for party pid, do:
$b_{\mathsf{vrfy}} := \mathsf{Vrfy}(pk, x, \sigma)$
if $pk = \mathsf{pk} \wedge \in \mathsf{H}_{\mathsf{sign}}[x]$**:** $b_{\mathsf{vrfy}} := 1$ { *Correctness*
else if $pk = \mathsf{pk} \wedge \exists (\widetilde{\sigma}, Y, y)$ s.t.
$(\widetilde{\sigma}, Y) \in \mathsf{H}_{\mathsf{pVerify}}[x] \wedge (\widetilde{\sigma}, Y, y, \sigma) \in \mathsf{H}_{\mathsf{adapt}}$**:** $b_{\mathsf{vrfy}} := 1$ { *Pre-Sig Adaptability*
Return $(\texttt{VerResult}, b_{\mathsf{vrfy}})$

As usual, the attacker may corrupt any party pid dynamically and gains full control over that party. In addition, if pid is the owner of the key pair modeled by this instance, i.e., the session ID of this instance is $sid = (pid, sid')$, then the attacker also learns sk; for notational convenience, we set $\mathsf{corr} = \mathsf{true}$ iff this is the case.

Fig. 12. Ideal adaptor signatures functionality $\mathcal{F}^2_{\text{asig}}$.

The functionality is parameterized by a polytime decidable relation $\mathcal{R}$, ppt algorithms $\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{pSign}$ and dpt algorithms $\mathsf{Vrfy}, \mathsf{pVrfy}, \mathsf{Adapt}, \mathsf{Extract}$. Upon first activation, initialize this instance by running and storing $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$.

On $\texttt{PubKey}$? from anyone, return $(\texttt{PubKey}, \mathsf{pk})$.

On $(\texttt{Sign}, x)$ for party pid in session $sid = (pid, sid')$, do:
- $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, x)$
- **if** $\sigma \neq \bot$**: add** σ **to** $\mathsf{H}_{\mathsf{sign}}[x]$
- Return $(\texttt{Signature}, \sigma)$.

On $(\texttt{pSign}, x, Y)$ for party pid in session $sid = (pid, sid')$, do:
- $\widetilde{\sigma} \leftarrow \mathsf{pSign}(\mathsf{sk}, x, Y)$
- **if** $\widetilde{\sigma} \neq \bot$**: add** $(\widetilde{\sigma}, Y)$ **to** $\mathsf{H}_{\mathsf{pSign}}[x]$
- Return $(\texttt{PreSignature}, \widetilde{\sigma})$

On $(\texttt{Adapt}, pk, \widetilde{\sigma}, y, Y)$ for party pid, do:
- $\sigma \leftarrow \mathsf{Adapt}(pk, \widetilde{\sigma}, y, Y)$
- **if** $pk = \mathsf{pk} \wedge (Y, y) \in \mathcal{R}$**: add** $(\widetilde{\sigma}, Y, y, \sigma)$ **to** $\mathsf{H}_{\mathsf{adapt}}$
- Return $(\texttt{Adapt}, \sigma)$

On $(\texttt{Extract}, pk, \widetilde{\sigma}, \sigma, Y)$ for party pid, do:
- $y := \mathsf{Extract}(pk, \widetilde{\sigma}, \sigma, Y)$
- **if** $(Y, y) \notin \mathcal{R} \wedge \exists (\widetilde{\sigma}, Y, y', \sigma) \in \mathsf{H}_{\mathsf{adapt}}$**:** $y := y'$ { *Extract Correctness*
- Return $(\texttt{Extract}, y)$

On $(\texttt{pVerify}, pk, x, Y, \widetilde{\sigma})$ for party pid, do:
- $b_{\mathsf{pVerify}} := \mathsf{pVrfy}(pk, x, Y, \widetilde{\sigma})$
- **if** $pk = \mathsf{pk} \wedge (\widetilde{\sigma}, Y) \in \mathsf{H}_{\mathsf{pSign}}[x]$**:** $b_{\mathsf{pVerify}} := 1$ { *Pre-Sign Correctness*
- **else if** $pk = \mathsf{pk} \wedge \neg\mathsf{corr} \wedge \mathsf{H}_{\mathsf{sign}}[x] = \emptyset$**:** $b_{\mathsf{pVerify}} := 0$ { *Pre-Sig Unforgeability*
- **if** $pk = \mathsf{pk} \wedge b_{\mathsf{pVerify}}$**: add** $(\widetilde{\sigma}, Y)$ $\mathsf{H}_{\mathsf{pVerify}}[x]$ **to**
- Return $(\texttt{VerResult}, b_{\mathsf{pVerify}})$

On $(\texttt{Verify}, pk, x, \sigma)$ for party pid, do:
- $b_{\mathsf{vrfy}} := \mathsf{Vrfy}(pk, x, \sigma)$
- **if** $pk = \mathsf{pk} \wedge \in \mathsf{H}_{\mathsf{sign}}[x]$**:** $b_{\mathsf{vrfy}} := 1$ { *Correctness*
- **else if** $pk = \mathsf{pk} \wedge \exists (\widetilde{\sigma}, Y, y)$ s.t.
  $(\widetilde{\sigma}, Y) \in \mathsf{H}_{\mathsf{pVerify}}[x] \wedge (\widetilde{\sigma}, Y, y, \sigma) \in \mathsf{H}_{\mathsf{adapt}}$**:** $b_{\mathsf{vrfy}} := 1$ { *Pre-Sig Adaptability*
- Return $(\texttt{VerResult}, b_{\mathsf{vrfy}})$

As usual, the attacker may corrupt any party pid dynamically and gains full control over that party. In addition, if pid is the owner of the key pair modeled by this instance, i.e., the session ID of this instance is $sid = (pid, sid')$, then the attacker also learns sk; for notational convenience, we set $\mathsf{corr} = \mathsf{true}$ iff this is the case.

Fig. 13. Ideal adaptor signatures functionality $\mathcal{F}_{\text{asig}}^3$.

provided public key. To simulate the functionality in the environment, our reduction uses this key and the provided oracles to answer signing and pre-signing queries. Adapting, extracting, and verifying are public algorithms, so our reduction can simulate them while applying the rules established in $\mathcal{F}^2_{\text{asig}}$. Since we only enforce unforgeable pre-signatures for uncorrupted signers, the reduction does not need to provide the signing key to the adversary. When the pre-verification oracle is called on a pre-signature–statement pair such that the pre-verification algorithm outputs 1, but the corresponding message was never signed nor pre-signed w.r.t. this statement, our reduction outputs this pre-signature as forgery. This forgery is valid, since it was not output by the pre-sign oracle, and the message x was never signed before. Since we just call efficient oracles polynomially often, our reduction is efficient. Whenever the functionality $\mathcal{F}^3_{\text{asig}}$ differs from $\mathcal{F}^2_{\text{asig}}$, our reduction breaks the pre-signature unforgeability of AS. As we assume AS to have unforgeable pre-signatures, the gap between $\mathcal{F}^3_{\text{asig}}$ and $\mathcal{F}^2_{\text{asig}}$ is negligible in the security parameter.

Game $\mathcal{G}_4$. In this game, we enforce extractability for an uncorrupted signer. This means, in this functionality, each signature that is valid either was previously signed or allows extracting a valid witness together with a pre-signature previously output by the pre-signing oracle. Besides this, the functionality $\mathcal{F}^4_{\text{asig}}$ has the same outputs as $\mathcal{F}^3_{\text{asig}}$. Furthermore, this functionality is also our final functionality. We depict $\mathcal{F}^4_{\text{asig}}$ in Figure 14.

Claim 4 *If AS, in addition to all the properties required by previous claims, satisfies extractability (Definition 10), then the gap between $\mathcal{F}^4_{asig}$ and $\mathcal{F}^3_{asig}$ is negligible.*

Proof. The outputs of the functionalities $\mathcal{F}^4_{\text{asig}}$ and $\mathcal{F}^3_{\text{asig}}$ differ only if the signer is uncorrupted, and the verification algorithm is called on a signature and a message x, such that the signature should be valid by the verification algorithm, but the message was never previously signed, i.e. $\mathsf{H}_{\mathsf{sign}}[x] = \emptyset$, and there exists no pre-signature–statement pair $(\widetilde{\sigma}, Y) \in \mathsf{H}_{\mathsf{pSign}}[x]$ such that the extract algorithm does not output a valid witness for Y when called using the candidate signature and $\widetilde{\sigma}$. In such cases, $\mathcal{F}^4_{\text{asig}}$ does not qualify the signature to be valid, while $\mathcal{F}^3_{\text{asig}}$ would do so. We now show that the probability of such cases occurring is negligible in the security parameter. To this end, we provide a reduction that transforms any environment causing this deviation into an adversary that breaks the extractability of AS. As input, our reduction receives a public key and has access to a signing and a pre-signing oracle that outputs valid (pre-)signatures w.r.t. the provided public key. To simulate the functionality in the environment, our reduction uses this key and the provided oracles to answer signing and pre-signing queries. Adapting, extracting, and verifying are public algorithms, so our reduction can simulate them while applying the rules established in $\mathcal{F}^3_{\text{asig}}$.Since we enforce extractability only for honest signers, the reduction does not need to provide a matching signing key to the adversary. When the environment provokes a gap between $\mathcal{F}^3_{\text{asig}}$ and $\mathcal{F}^4_{\text{asig}}$, our reduction has available a signature–message pair such that the verification algorithm verifies, and there exists no pre-signature that qualifies for extracting a valid witness using the $\mathsf{Extract}$ algorithm, and the message was never signed before.

Since we just call efficient oracles and compute efficient algorithms polynomially often, our reduction is efficient. Whenever the functionality $\mathcal{F}^4_{\text{asig}}$ differs from $\mathcal{F}^3_{\text{asig}}$, our reduction breaks the extractability of AS. As we assume AS to have extractability, the gap between $\mathcal{F}^4_{\text{asig}}$ and $\mathcal{F}^3_{\text{asig}}$ is negligible in the security parameter.

D Extended Functionality

In this section, we provide the extended adaptor signature ideal functionality, which, in addition to correctness, adaptability, and extractability, also achieves unique extractability. We refer to this extended functionality as $\mathcal{F}^{(\text{ext})}_{\text{asig}}$. As we have already pointed out, such extended functionality requires resolving another issue to reconcile unique extractability and achieve $\mathcal{F}^{(\text{ext})}_{\text{asig}}$. We therefore require message-bound pre-signatures.

Message-bound pre-signatures. An adaptor signature scheme has *message-bound pre-signatures* if any given pre-signature is valid for at most one message, except with negligible probability. The concept of message-bound signatures is well-established for ordinary signatures [SPMS02,JCCS19,BCJZ21,CDF+21]; here, we adapt and refine this idea in the context of pre-signatures.

The functionality is parameterized by a polytime decidable relation $\mathcal{R}$, ppt algorithms $\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{pSign}$ and dpt algorithms $\mathsf{Vrfy}, \mathsf{pVrfy}, \mathsf{Adapt}, \mathsf{Extract}$. Upon first activation, initialize this instance by running and storing $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$.

On $\mathtt{PubKey}$? from anyone, return $(\mathtt{PubKey}, \mathsf{pk})$.

On $(\mathtt{Sign}, x)$ for party pid in session $sid = (pid, sid')$, do:
$\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, x)$
if $\sigma \neq \bot$**: add** σ **to** $\mathsf{H}_{\mathsf{sign}}[x]$
Return $(\mathtt{Signature}, \sigma)$.

On $(\mathtt{pSign}, x, Y)$ for party pid in session $sid = (pid, sid')$, do:
$\widetilde{\sigma} \leftarrow \mathsf{pSign}(\mathsf{sk}, x, Y)$
if $\widetilde{\sigma} \neq \bot$**: add** $(\widetilde{\sigma}, Y)$ **to** $\mathsf{H}_{\mathsf{pSign}}[x]$
Return $(\mathtt{PreSignature}, \widetilde{\sigma})$

On $(\mathtt{Adapt}, pk, \widetilde{\sigma}, y, Y)$ for party pid, do:
$\sigma \leftarrow \mathsf{Adapt}(pk, \widetilde{\sigma}, y, Y)$
if $pk = \mathsf{pk} \wedge (Y, y) \in \mathcal{R}$**: add** $(\widetilde{\sigma}, Y, y, \sigma)$ **to** $\mathsf{H}_{\mathsf{adapt}}$
Return $(\mathtt{Adapt}, \sigma)$

On $(\mathtt{Extract}, pk, \widetilde{\sigma}, \sigma, Y)$ for party pid, do:
$y := \mathsf{Extract}(pk, \widetilde{\sigma}, \sigma, Y)$
if $(Y, y) \notin \mathcal{R} \wedge \exists (\widetilde{\sigma}, Y, y', \sigma) \in \mathsf{H}_{\mathsf{adapt}}$**:** $y := y'$ { *Extract Correctness*
Return $(\mathtt{Extract}, y)$

On $(\mathtt{pVerify}, pk, x, Y, \widetilde{\sigma})$ for party pid, do:
$b_{\mathsf{pVerify}} := \mathsf{pVrfy}(pk, x, Y, \widetilde{\sigma})$
if $pk = \mathsf{pk} \wedge (\widetilde{\sigma}, Y) \in \mathsf{H}_{\mathsf{pSign}}[x]$**:** $b_{\mathsf{pVerify}} := 1$ { *Pre-Sign Correctness*
else if $pk = \mathsf{pk} \wedge \neg\mathsf{corr} \wedge \mathsf{H}_{\mathsf{sign}}[x] = \emptyset$**:** $b_{\mathsf{pVerify}} := 0$ { *Pre-Sig Unforgeability*
if $pk = \mathsf{pk} \wedge b_{\mathsf{pVerify}}$**: add** $(\widetilde{\sigma}, Y)$ $\mathsf{H}_{\mathsf{pVerify}}[x]$ **to**
Return $(\mathtt{VerResult}, b_{\mathsf{pVerify}})$

On $(\mathtt{Verify}, pk, x, \sigma)$ for party pid, do:
$b_{\mathsf{vrfy}} := \mathsf{Vrfy}(pk, x, \sigma)$
$b_{\mathsf{ext}} := \exists (\widetilde{\sigma}, Y) \in \mathsf{H}_{\mathsf{pSign}}[x] : (Y, \mathsf{Extract}(pk, \widetilde{\sigma}, \sigma, Y)) \in \mathcal{R}$
if $pk = \mathsf{pk} \wedge \in \mathsf{H}_{\mathsf{sign}}[x]$**:** $b_{\mathsf{vrfy}} := 1$ { *Correctness*
else if $pk = \mathsf{pk} \wedge \exists (\widetilde{\sigma}, Y, y)$ s.t.
$(\widetilde{\sigma}, Y) \in \mathsf{H}_{\mathsf{pVerify}}[x] \wedge (\widetilde{\sigma}, Y, y, \sigma) \in \mathsf{H}_{\mathsf{adapt}}$**:** $b_{\mathsf{vrfy}} := 1$ { *Pre-Sig Adaptability*
else if $pk = \mathsf{pk} \wedge \neg\mathsf{corr} \wedge \mathsf{H}_{\mathsf{sign}}[x] = \emptyset \wedge \neg b_{\mathsf{ext}}$**:** $b_{\mathsf{vrfy}} := 0$ { *Extractability*
Return $(\mathtt{VerResult}, b_{\mathsf{vrfy}})$

As usual, the attacker may corrupt any party pid dynamically and gains full control over that party. In addition, if pid is the owner of the key pair modeled by this instance, i.e., the session ID of this instance is $sid = (pid, sid')$, then the attacker also learns sk; for notational convenience, we set $\mathsf{corr} = \mathsf{true}$ iff this is the case.

Fig. 14. Ideal adaptor signatures functionality $\mathcal{F}_{\mathrm{asig}}^4$.

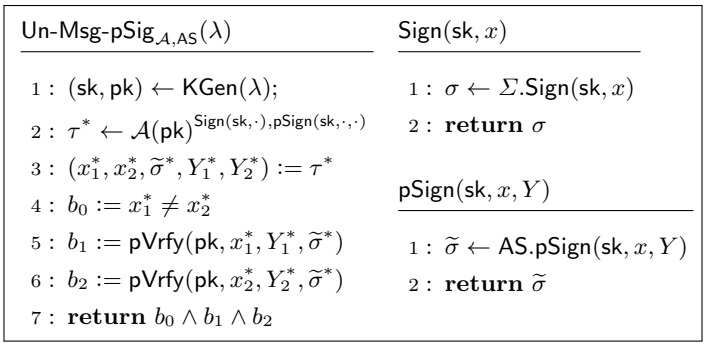

Fig. 15. The message-bound pre-signature experiment Un-Msg-pSig$_{\mathcal{A},\mathsf{AS}}(\lambda)$.

The underlying issue that message-bound pre-signatures resolve is that the Adapt and Extract algorithms, and hence the corresponding oracles in an ideal functionality, do not take the original message that was pre-signed as input. Indeed, at the time of adaptation, the adapting party might be entirely unaware of the original message. However, information about the underlying message is needed in the extended ideal functionality to enforce the unique extractability property during adaptation, since this property depends on the (pre-)signed message. By requiring message-bound pre-signatures (i.e., each pre-signature verifying at most one unique message), the ideal functionality is able to look up the corresponding message in its verification history (cf. Section 4).

Definition 13 (Message-Bound Pre-Signatures). *An adaptor signature scheme* AS *has* message-bound pre-signatures *if for every* PPT *adversary* $\mathcal{A}$*, there exists a negligible function* negl *such that for every* $\lambda \in \mathbb{N}$*,*

$$\Pr\left[\mathsf{Un\text{-}Msg\text{-}pSig}_{\mathcal{A},\mathsf{AS}}(\lambda) = 1\right] \leq \mathsf{negl}(\lambda) \ ,$$

where the experiment Un-Msg-pSig$_{\mathcal{A},\mathsf{AS}}$ *is defined in Figure 15, and the probability is taken over the randomness of all probabilistic algorithms.*

With message-bound pre-signatures, we can now describe the additional steps needed to prove the security of $\mathcal{F}_{\text{asig}}$. These steps need to be added to the proof steps of $\mathcal{F}_{\text{asig}}$. After correctness and prior to pre-signature adaptability, we can add message-bound pre-signatures.

Adding property: message-bound pre-signatures. Once established correctness, we enforce that each pre-signature is valid for exactly one message. As mentioned above, while this is not implied by the key security properties that we want $\mathcal{F}_{\text{asig}}$ to formalize, it appears to be a necessary additional requirement to reconcile pre-signature adaptability and unique extractability in the definition of a composable ideal functionality.

To enforce message-bound pre-signatures, the functionality checks during pre-signing and pre-verification whether a candidate pre-signature $\widetilde{\sigma}$ has already been accepted as valid for a different message during pre-signing or pre-verification, i.e., already is in the sets $\mathsf{H}_{\mathsf{pSign}}$ or $\mathsf{H}_{\mathsf{pVerify}}$. If so, the pre-signing oracle instead returns an error $\perp$, and the pre-verification oracle returns false. Observe that the negative *message-bound pre-signature* property does not clash with the positive *correctness* property due to how we change the pSign oracle: pre-signature creation is successful and hence will add the message and pre-signature to $\mathsf{H}_{\mathsf{pSign}}$ only if the pre-signature does not violate message-boundedness. But from then on, the pre-signature is bound to that message, i.e., will also not violate message-boundedness during verification w.r.t. the original message. One can thus safely define verification to always succeed in this case.

We depict this enhanced version of the functionality in Figure 19 with changes marked in red, and prove in Claim 6 that any adaptor signature scheme AS with message-bound pre-signatures (cf. Definition 13) realizes this version.

Once message-bound pre-signatures are in place, and prior to enforcing strongly unforgeable pre-signatures, we enforce unique extractability.

Adding property: unique extractability. Next, we extend the functionality to guarantee *unique extractability*. Recall that this property requires that for any given pre-signature $\widetilde{\sigma}$ which is valid for a message x and statement Y, there exists at most one valid full signature σ on the same message x such that the pair $(\widetilde{\sigma}, \sigma)$ enables extraction of a valid witness for statement Y. Our functionality must enforce this condition at all points where new valid (pre-)signatures are introduced, since those may later be used in verification or extraction. Concretely, this situation arises in three cases:

- **During signing and pre-signing:** (Pre-)signatures generated by oracles are automatically considered valid due to correctness by the previous step. Thus, (pre-)signature generation must not output (pre-)signatures that would violate unique extractability.
- **During (pre-)verification:** any new (pre-)signature violating unique extractability must not (pre-)verify.
- **During honest adaptation:** Due to pre-signature adaptability, we define all full signatures contained in $\mathsf{H}_{\mathsf{adapt}}$ to always be considered valid. Thus, Adapt will then be able to introduce new valid signatures, and we have to change the oracle now to ensure those signatures added to $\mathsf{H}_{\mathsf{adapt}}$ do not violate unique extractability.

We now explain how we enforce unique extractability. When a new pre-signature $\widetilde{\sigma}$ on some message x and statement Y appears (either because the pSign oracle internally computed $\widetilde{\sigma}$ or because pVrfy was called with $\widetilde{\sigma}$ given as input) the functionality checks whether there already exist *two distinct valid signatures* $\sigma_1 \neq \sigma_2$ for x such that both $(\widetilde{\sigma}, \sigma_1)$ and $(\widetilde{\sigma}, \sigma_2)$ successfully extract (possibly different) witnesses for Y. Checking this involves considering the lists $\mathsf{H}_{\mathsf{sign}}$ and $\mathsf{H}_{\mathsf{verify}}$, which contain all valid signatures seen so far, both due to the correctness of signing and also due to past calls to the Vrfy oracle. If such a pair of distinct signatures is found, the pre-signature $\widetilde{\sigma}$ needs to be rejected by the functionality: the pSign oracle is changed to instead return an error, and the pVrfy oracle instead returns false. Note that this change does not clash with the correctness property of pVrfy because we have modified pSign as well: a pre-signature is added to $\mathsf{H}_{\mathsf{pSign}}$ only if it passes the unique extractability check during pre-signing, so it can still safely be accepted by pVrfy. Since this consistency check must be performed both during pre-signing and pre-verification, we encapsulate it in the helper method BreakpSigUnExt, depicted in Figure 17.

A similar situation arises when a new signature σ on some message x appears - during a call to the Sign, Vrfy, or Adapt oracles. In this case, the functionality must ensure that unique extractability is preserved with respect to existing valid pre-signatures for the same message x. Specifically, it checks whether there exists a valid pre-signature $\widetilde{\sigma}$ on x and some statement Y (stored in $\mathsf{H}_{\mathsf{pSign}}$ or $\mathsf{H}_{\mathsf{pVerify}}$) and a valid signature σ' on x (stored in $\mathsf{H}_{\mathsf{sign}}$, $\mathsf{H}_{\mathsf{verify}}$, or $\mathsf{H}_{\mathsf{adapt}}$) such that the Extract oracle would return witnesses for Y from both $(\widetilde{\sigma}, \sigma)$ and $(\widetilde{\sigma}, \sigma')$. If such a conflicting signature σ' exists, the functionality takes two actions: the Sign oracle returns an error, consistently with the pSign oracle. The Adapt is modified to instead return the already known extracting signature σ'. Both actions effectively prevent the introduction of non-uniquely extracting signatures. At the same time, the Vrfy oracle is modified to reject σ, ensuring that at most one valid extracting signature is accepted per pre-signature. We capture this behavior in the helper function BreakSigUnExt, shown in Figure 17.

Let us point out several important details: First, signature correctness is again compatible with the changes to Vrfy by the same reasoning as for pre-signature correctness. Second, while for pSign and Sign we can simply return errors in the case where a conflicting (pre-)signature would be introduced, the Adapt oracle has to follow the reprogramming approach. This is the case since, for pre-signature adaptability, we have to ensure that Adapt, when given correct inputs, will return a signature. Third, to run the BreakSigUnExt check in Adapt, Adapt needs access to the underlying message x, which we can obtain from the input pre-signature thanks to message-boundedness introduced in a previous step. Fourth, since we have changed the Extract oracle to sometimes return witnesses when the underlying Extract algorithm might not, namely whenever pre-signature and signature are in the set $\mathsf{H}_{\mathsf{adapt}}$, the helper function BreakSigUnExt needs to be a bit more involved and take this modification into account. In contrast, BreakpSigUnExt can be simpler since the Extract oracle differs from the Extract algorithm only if the pre-signature $\widetilde{\sigma}$ being checked by BreakpSigUnExt is in $\mathsf{H}_{\mathsf{adapt}}$, in which case $\widetilde{\sigma}$ is not new but has already previously been verified to not violate unique extractability.

We show in Claim 8 that an adaptor signature scheme that is uniquely extractable, in addition to the already required properties, satisfies the enhanced intermediate functionality (Figure 21).

These are the two major changes compared to the functionality $\mathcal{F}_{\text{asig}}$. Note that in addition, we handle pre-signature adaptability slightly differently, since we only consider adding pre-signatures to the list $\mathsf{H}_{\mathsf{adapt}}$ if the corresponding pre-signature pre-verifies. This can be checked in $\mathcal{F}_{\text{asig}}^{(\text{ext})}$, but not in $\mathcal{F}_{\text{asig}}$, since in $\mathcal{F}_{\text{asig}}$, the adapt algorithm is not aware of the unique message x, w.r.t. which a pre-signature can be valid. We depict the final functionality in Figure 16. To remove redundancy, we provide the helper functions in Figure 17.

D.1 Realization of $\mathcal{F}_{\text{asig}}^{(\text{ext})}$

Theorem 4. *Let* $\mathsf{AS}_{\Sigma,\mathcal{R}} = (\mathsf{pSign}, \mathsf{pVrfy}, \mathsf{Adapt}, \mathsf{Extract})$ *be an adaptor signature scheme for a correct signature scheme* $\Sigma = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vf})$ *and a hard relation* $\mathcal{R}$*. Let* $\mathcal{F}_{asig}^{(ext)}$ *be parameterized with the same algorithms. If* $\mathsf{AS}_{\Sigma,\mathcal{R}}$ *is extractable, pre-signature adaptable, uniquely extractable, correct, and has unforgeable and message-bound pre-signatures, then*

$$\mathsf{AS}_{\Sigma,\mathcal{R}} \leq \mathcal{F}_{asig}^{(ext)}.$$

Proof (Proof of Theorem 4). Similar to the proof of Theorem 2, we prove Theorem 4 by showing the existence of a simulator $\mathcal{S}$ such that the real world protocol that just runs $\mathsf{AS}_{\Sigma,\mathcal{R}}$ (in the presence of the dummy adversary that simply forwards network messages) is indistinguishable from the ideal functionality $\mathcal{F}_{\text{asig}}^{(\text{ext})}$ running with the simulator $\mathcal{S}$ for all ppt environments. As a starting point, we again use the realization functionality Figure 2.

Game $\mathcal{G}_1$. In the first game, we start by modifying our real protocol to obtain a first intermediate ideal functionality that enforces correctness properties in the verification, pre-verification, and extract oracles. To this extent, our functionality deploys bookkeeping. This means, when a signature or pre-signature is successfully created, the functionality stores the corresponding (pre-)signatures into lists $\mathsf{H}_{\mathsf{sign}}$ and $\mathsf{H}_{\mathsf{pSign}}$. When a signature or pre-signature is verified, the functionality accepts all signatures from $\mathsf{H}_{\mathsf{sign}}$ and all pre-signatures from $\mathsf{H}_{\mathsf{pSign}}$. In addition, the functionality stores the results of successfully (pre-)verifying (pre-)signatures in $\mathsf{H}_{\mathsf{verify}}$ and $\mathsf{H}_{\mathsf{pVerify}}$. As an additional list, we introduce the list $\mathsf{H}_{\mathsf{adapt}}$. In $\mathsf{H}_{\mathsf{adapt}}$, we store the inputs and the output of the adapt oracle, i.e., the pairs $(\widetilde{\sigma}, Y, y, \sigma)$, when the pre-signature–statement pair is stored in $\mathsf{H}_{\mathsf{pVerify}}$, the witness y is a valid witness for Y, and the signature σ is the output of calling the adapt algorithm on $(\widetilde{\sigma}, Y, y)$. Looking ahead, this list will allow us to enforce extract correctness, as well as pre-signature adaptability. Besides this, the functionality $\mathcal{F}_{\text{asig}}^{(\text{ext},1)}$ has the same outputs as $\mathcal{F}_{\text{asig}}^{(\text{ext},0)}$. We depict $\mathcal{F}_{\text{asig}}^{(\text{ext},1)}$ in Figure 18.

Claim 5 *If* AS *is pre-sign correct (Definition 7), has correct extracting (Definition 8), and* Σ *is a correct signature scheme (Definition 4), then the gap between* $\mathcal{F}_{asig}^{(ext,1)}$ *and* $\mathcal{F}_{asig}^{(ext,0)}$ *is negligible.*

Proof. To show Claim 5, we note that $\mathcal{F}_{\text{asig}}^{(\text{ext},1)}$ and $\mathcal{F}_{\text{asig}}^{(\text{ext},0)}$ differ in three cases. The first case is, if a signature on a message x was obtained via $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, x)$ by an oracle, and $\mathsf{Vrfy}(\mathsf{pk}, x, \sigma) = 0$. As the signature scheme Σ is (perfectly) correct, $\mathsf{Vrfy}(\mathsf{pk}, x, \mathsf{Sign}(\mathsf{sk}, x)) = 0$ cannot happen. The second case is equivalent to the first, but when pre-verifying pre-signatures output by the pre-sign oracle. This second change is again undetectable by the perfect pre-sign correctness of the adaptor signature scheme. In the third case, the functionalities differ if the extract algorithm outputs bot on a pre-signature–signature–statement pair, but this pair is added to $\mathsf{H}_{\mathsf{adapt}}$. A pre-signature–signature–statement pair is added to $\mathsf{H}_{\mathsf{adapt}}$ only if the pre-signature–statement pair was previously stored in $\mathsf{H}_{\mathsf{pVerify}}$. This occurs only if the pre-signature verifies (for that statement and some message x) under the pVrfy algorithm or if it was stored (for that statement and some message x) in $\mathsf{H}_{\mathsf{pSign}}$. By pre-sign correctness, all entries in $\mathsf{H}_{\mathsf{pSign}}$ also verify under the pVrfy algorithm. Altogether we have that in all cases where an element is added to $\mathsf{H}_{\mathsf{adapt}}$ the pre-signature for the statement is valid under the pVrfy algorithm (for some message x). In addition to such a valid pre-signature, adding a pair to $\mathsf{H}_{\mathsf{adapt}}$ also requires providing a valid witness for the statement. Only if a valid pre-signature–statement pair and a valid witness for this statement are provided, we call the adapt algorithm on this input and store the valid pre-signature, the statement, the valid witness, and the resulting adapted pre-signature in $\mathsf{H}_{\mathsf{adapt}}$. When the functionality overwrites a witness, the extract algorithm failed to provide a valid witness for a valid

The functionality is parameterized by a polytime decidable relation $\mathcal{R}$, ppt algorithms $\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{pSign}$ and dpt algorithms $\mathsf{Vrfy}, \mathsf{pVrfy}, \mathsf{Adapt}, \mathsf{Extract}$. Upon first activation, initialize this instance by running and storing $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$.

On $\mathtt{PubKey}$? from anyone, return $(\mathtt{PubKey}, \mathsf{pk})$.

On $(\mathtt{Sign}, x)$ for party pid in session $sid = (pid, sid')$, do:
- $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, x)$
- **if** $\mathsf{BreakSigUnExt}(x, \sigma) \neq \bot$**:** $\sigma := \bot$ {*Unique Extractability*
- **if** $\sigma \neq \bot$**: add** σ **to** $\mathsf{H}_{\mathsf{sign}}[x]$
- Return $(\mathtt{Signature}, \sigma)$.

On $(\mathtt{pSign}, x, Y)$ for party pid in session $sid = (pid, sid')$, do:
- $\widetilde{\sigma} \leftarrow \mathsf{pSign}(\mathsf{sk}, x, Y)$
- **if** $\neg\mathsf{corr} \wedge \exists x' \neq x : \widetilde{\sigma} \in \mathsf{H}_{\mathsf{pSign}}[x'] \cup \mathsf{H}_{\mathsf{pVerify}}[x']$**:** $\widetilde{\sigma} := \bot$ {*Message Bounding*
- **if** $\mathsf{BreakpSigUnExt}(x, \widetilde{\sigma}, Y) \neq \bot$**:** $\widetilde{\sigma} := \bot$ {*Unique Extractability*
- **if** $\widetilde{\sigma} \neq \bot$**: add** $(\widetilde{\sigma}, Y)$ **to** $\mathsf{H}_{\mathsf{pSign}}[x]$
- Return $(\mathtt{PreSignature}, \widetilde{\sigma})$

On $(\mathtt{Adapt}, pk, \widetilde{\sigma}, y, Y)$ for party pid, do:
- $\sigma \leftarrow \mathsf{Adapt}(pk, \widetilde{\sigma}, y, Y)$
- **if** $pk = \mathsf{pk} \wedge (Y, y) \in \mathcal{R} \wedge \exists x : (\widetilde{\sigma}, Y) \in \mathsf{H}_{\mathsf{pVerify}}[x]$**:**
  - **if** $\sigma' := \mathsf{BreakSigUnExt}(x, \sigma) \wedge \sigma' \neq \bot$**:** $\sigma \leftarrow \sigma'$ {*Unique Extractability*
  - **add** $(\widetilde{\sigma}, Y, y, \sigma)$ **to** $\mathsf{H}_{\mathsf{adapt}}[x]$
- Return $(\mathtt{Adapt}, \sigma)$

On $(\mathtt{Extract}, pk, \widetilde{\sigma}, \sigma, Y)$ for party pid, do:
- $y := \mathsf{Extract}(pk, \widetilde{\sigma}, \sigma, Y)$
- **if** $(Y, y) \notin \mathcal{R} \wedge \exists (x, y') : (\widetilde{\sigma}, Y, y', \sigma) \in \mathsf{H}_{\mathsf{adapt}}[x]$**:** $y := y'$ {*Extract Correctness*
- Return $(\mathtt{Extract}, y)$

On $(\mathtt{pVerify}, pk, x, Y, \widetilde{\sigma})$ for party pid, do:
- $b_{\mathsf{pVerify}} := \mathsf{pVrfy}(pk, x, Y, \widetilde{\sigma})$
- **if** $pk = \mathsf{pk} \wedge (\widetilde{\sigma}, Y) \in \mathsf{H}_{\mathsf{pSign}}[x]$**:** $b_{\mathsf{pVerify}} := 1$ {*Pre-Sign Correctness*
- **else if** $pk = \mathsf{pk} \wedge \neg\mathsf{corr}$**:**
  - **if** $\exists x' \neq x : \widetilde{\sigma} \in \mathsf{H}_{\mathsf{pSign}}[x'] \cup \mathsf{H}_{\mathsf{pVerify}}[x']$**:** $b_{\mathsf{pVerify}} := 0$ {*Message Bounding*
    - **if** $\mathsf{BreakpSigUnExt}(x, \widetilde{\sigma}) \neq \bot$**:** $b_{\mathsf{vrfy}} := 0$ {*Unique Extractability*
    - **if** $\mathsf{H}_{\mathsf{sign}}[x] = \emptyset$**:** $b_{\mathsf{pVerify}} := 0$ {*Pre-Sig Unforgeability*
- **if** $pk = \mathsf{pk} \wedge b_{\mathsf{pVerify}}$**: add** $(\widetilde{\sigma}, Y)$ **to** $\mathsf{H}_{\mathsf{pVerify}}[x]$
- Return $(\mathtt{VerResult}, b_{\mathsf{pVerify}})$

On $(\mathtt{Verify}, pk, x, \sigma)$ for party pid, do:
- $b_{\mathsf{vrfy}} := \mathsf{Vrfy}(pk, x, \sigma)$
- $b_{\mathsf{ext}} := \exists (\widetilde{\sigma}, Y) \in \mathsf{H}_{\mathsf{pSign}}[x] : (Y, \mathsf{Extract}(pk, \widetilde{\sigma}, \sigma, Y)) \in \mathcal{R}$
- **if** $pk = \mathsf{pk} \wedge (x, \sigma) \in \mathsf{H}_{\mathsf{sign}}$**:** $b_{\mathsf{vrfy}} := 1$ {*Correctness*
- **else if** $pk = \mathsf{pk} \wedge \exists (\widetilde{\sigma}, Y, y, \sigma) \in \mathsf{H}_{\mathsf{adapt}}[x]$**:** $b_{\mathsf{vrfy}} := 1$ {*Pre-Sig Adaptability*
- **else if** $pk = \mathsf{pk} \wedge \neg\mathsf{corr}$**:**
  - **if** $\mathsf{BreakSigUnExt}(x, \sigma) \neq \bot$**:** $b_{\mathsf{vrfy}} := 0$ {*Unique Extractability*
  - **if** $\mathsf{H}_{\mathsf{sign}}[x] = \emptyset \wedge \neg b_{\mathsf{ext}}$**:** $b_{\mathsf{vrfy}} := 0$ {*Extractability*
- **if** $pk = \mathsf{pk} \wedge b_{\mathsf{vrfy}}$**: add** σ **to** $\mathsf{H}_{\mathsf{verify}}[x]$
- Return $(\mathtt{VerResult}, b_{\mathsf{vrfy}})$

As usual, the attacker may corrupt any party pid dynamically and gains full control over that party. In addition, if pid is the owner of the key pair modeled by this instance, i.e., the session ID of this instance is $sid = (pid, sid')$, then the attacker also learns sk; for notational convenience, we set $\mathsf{corr} = \mathsf{true}$ iff this is the case.

Fig. 16. Extended ideal adaptor signatures functionality $\mathcal{F}_{\mathrm{asig}}^{(\mathrm{ext})}$.

$\mathsf{BreakSigUnExt}(x, \sigma)$

1 : **if** corr : **return** $\bot$
2 : $\mathcal{Q} \leftarrow \{(\widetilde{\sigma}, Y) | (\widetilde{\sigma}, Y) \in \mathsf{H}_{\mathsf{pVerify}}[x] \cup \mathsf{H}_{\mathsf{pSign}}[x] : (Y, \mathsf{Extract}(\mathsf{pk}, \widetilde{\sigma}, \sigma, Y)) \in \mathcal{R}\}$
3 : **if** $\exists(\widetilde{\sigma}, Y, \cdot, \sigma') \in \mathsf{H}_{\mathsf{adapt}}[x] : (\sigma' \neq \sigma \wedge (\widetilde{\sigma}, Y) \in \mathcal{Q})$
4 : **return** σ'
5 : **if** $\exists \sigma' \neq \sigma \in \mathsf{H}_{\mathsf{sign}}[x] \cup \mathsf{H}_{\mathsf{verify}}[x], (\widetilde{\sigma}, Y) \in \mathcal{Q} : (Y, \mathsf{Extract}(\mathsf{pk}, \widetilde{\sigma}, \sigma', Y)) \in \mathcal{R}$
6 : **return** σ'
7 : **return** $\bot$

$\mathsf{BreakpSigUnExt}(x, \widetilde{\sigma}, Y)$

1 : **if** corr : **return** $\bot$
2 : $\mathcal{Q} \leftarrow \{\sigma | \sigma \in \mathsf{H}_{\mathsf{verify}}[x] \cup \mathsf{H}_{\mathsf{sign}}[x] : (Y, \mathsf{Extract}(\mathsf{pk}, \widetilde{\sigma}, \sigma, Y)) \in \mathcal{R}\}$
3 : **if** $|\mathcal{Q}| > 1$: **return** $\top$
4 : **return** $\bot$

Fig. 17. Helper Functions for Unique Extractability.

pre-signature, statement, signature pair. If this case occurs, it would thus contradict extract correctness. Since extract correctness is a perfect property, this case cannot happen.

Hence, the gap between $\mathcal{F}_{\text{asig}}^{(\text{ext},1)}$ and $\mathcal{F}_{\text{asig}}^{(\text{ext},0)}$ is bounded by correctness, pre-sign correctness, and adapting correctness. Since these are perfect properties, this gap is 0.

At this point in our series of game hops, whenever an oracle considers a pre-signature or signature valid w.r.t. some message (and some statement), then the actual algorithms output that the (pre-)signature is valid. Hence, all signatures in the sets $\mathsf{H}_{\mathsf{sign}}[x], \mathsf{H}_{\mathsf{verify}}[x]$ will verify w.r.t. pk and x, and all pre-signature–statement pairs $(\widetilde{\sigma}, Y)$ in the sets $\mathsf{H}_{\mathsf{pSign}}[x], \mathsf{H}_{\mathsf{pVerify}}[x], \mathsf{H}_{\mathsf{adapt}}[x]$ will pre-verify w.r.t. x. This holds, since from this point forward, we will not add more elements to these lists, and assuming correctness and pre-sign correctness, which are perfect properties.

Game $\mathcal{G}_2$. In this game, the functionality enforces message-bound pre-signatures if the adversary did not corrupt the signer. This means the functionality enforces that each pre-signature can only be valid on a unique message. This ensures unique messages in the list $\mathsf{H}_{\mathsf{adapt}}$ when no corruption happened. Besides this, the functionality $\mathcal{F}_{\text{asig}}^{(\text{ext},2)}$ has the same outputs as $\mathcal{F}_{\text{asig}}^{(\text{ext},1)}$. We depict $\mathcal{F}_{\text{asig}}^{(\text{ext},2)}$ in Figure 19.

Claim 6 *If* AS*, in addition to all the properties required by previous claims, has message-bound pre-signatures (Definition 13), then the gap between* $\mathcal{F}_{asig}^{(ext,2)}$ *and* $\mathcal{F}_{asig}^{(ext,1)}$ *is negligible.*

Proof. The outputs of the functionalities $\mathcal{F}_{\text{asig}}^{(\text{ext},2)}$ and $\mathcal{F}_{\text{asig}}^{(\text{ext},1)}$ differ if the signer is uncorrupted, and the pre-signing algorithm pSign, on input a message x, outputs a pre-signature that is also considered valid with respect to a different message x', or if the environment provides a pre-signature that is valid with respect to the public key pk, a message x, and some statement Y, but this pre-signature has already been marked as valid for another message x'. In such cases, $\mathcal{F}_{\text{asig}}^{(\text{ext},2)}$ either returns $\bot$ instead of a pre-signature or fails to pre-verify a valid pre-signature.

We now show that the probability of this case occurring is negligible in the security parameter. To this end, we provide a reduction that transforms any environment causing a pre-signature that is valid on two different messages into an adversary that breaks the message-bound pre-signature property of AS.

As input, our reduction receives a public key and has access to a signing and a pre-signing oracle that outputs valid (pre-) signatures w.r.t. the provided public key. To simulate the functionality in the environment, our reduction uses this key and the provided oracles to answer signing and pre-signing queries. Adapting,

The functionality is parameterized by a polytime decidable relation $\mathcal{R}$, ppt algorithms $\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{pSign}$ and dpt algorithms $\mathsf{Vrfy}, \mathsf{pVrfy}, \mathsf{Adapt}, \mathsf{Extract}$. Upon first activation, initialize this instance by running and storing $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$.

On $\texttt{PubKey}$? from anyone, return $(\texttt{PubKey}, \mathsf{pk})$.

On $(\texttt{Sign}, x)$ for party pid in session $sid = (pid, sid')$, do:

$\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, x)$
if $\sigma \neq \bot$: **add** σ **to** $\mathsf{H}_{\mathsf{sign}}[x]$
Return $(\texttt{Signature}, \sigma)$.

On $(\texttt{pSign}, x, Y)$ for party pid in session $sid = (pid, sid')$, do:

$\widetilde{\sigma} \leftarrow \mathsf{pSign}(\mathsf{sk}, x, Y)$
if $\widetilde{\sigma} \neq \bot$: **add** $(\widetilde{\sigma}, Y)$ **to** $\mathsf{H}_{\mathsf{pSign}}[x]$
Return $(\texttt{PreSignature}, \widetilde{\sigma})$

On $(\texttt{Adapt}, pk, \widetilde{\sigma}, y, Y)$ for party pid, do:

$\sigma \leftarrow \mathsf{Adapt}(pk, \widetilde{\sigma}, y, Y)$
if $pk = \mathsf{pk} \wedge (Y, y) \in \mathcal{R} \wedge \exists x : (\widetilde{\sigma}, Y) \in \mathsf{H}_{\mathsf{pVerify}}[x]$:
add $(\widetilde{\sigma}, Y, y, \sigma)$ **to** $\mathsf{H}_{\mathsf{adapt}}[x]$
Return $(\texttt{Adapt}, \sigma)$

On $(\texttt{Extract}, pk, \widetilde{\sigma}, \sigma, Y)$ for party pid, do:

$y := \mathsf{Extract}(pk, \widetilde{\sigma}, \sigma, Y)$
if $(Y, y) \notin \mathcal{R} \wedge \exists (x, y') : (\widetilde{\sigma}, Y, y', \sigma) \in \mathsf{H}_{\mathsf{adapt}}[x]$: $y := y'$ {*Extract Correctness*
Return $(\texttt{Extract}, y)$

On $(\texttt{pVerify}, pk, x, Y, \widetilde{\sigma})$ for party pid, do:

$b_{\mathsf{pVerify}} := \mathsf{pVrfy}(pk, x, Y, \widetilde{\sigma})$
if $pk = \mathsf{pk} \wedge (\widetilde{\sigma}, Y) \in \mathsf{H}_{\mathsf{pSign}}[x]$: $b_{\mathsf{pVerify}} := 1$ {*Pre-Sign Correctness*
if $pk = \mathsf{pk} \wedge b_{\mathsf{pVerify}}$: **add** $(\widetilde{\sigma}, Y)$ **to** $\mathsf{H}_{\mathsf{pVerify}}[x]$
Return $(\texttt{VerResult}, b_{\mathsf{pVerify}})$

On $(\texttt{Verify}, pk, x, \sigma)$ for party pid, do:

$b_{\mathsf{vrfy}} := \mathsf{Vrfy}(pk, x, \sigma)$
if $pk = \mathsf{pk} \wedge (x, \sigma) \in \mathsf{H}_{\mathsf{sign}}$: $b_{\mathsf{vrfy}} := 1$ {*Correctness*
if $pk = \mathsf{pk} \wedge b_{\mathsf{vrfy}}$: **add** σ **to** $\mathsf{H}_{\mathsf{verify}}[x]$
Return $(\texttt{VerResult}, b_{\mathsf{vrfy}})$

As usual, the attacker may corrupt any party pid dynamically and gains full control over that party. In addition, if pid is the owner of the key pair modeled by this instance, i.e., the session ID of this instance is $sid = (pid, sid')$, then the attacker also learns sk; for notational convenience, we set $\mathsf{corr} = \mathsf{true}$ iff this is the case.

Fig. 18. Ideal adaptor signatures functionality $\mathcal{F}_{\mathrm{asig}}^1$.

The functionality is parameterized by a polytime decidable relation $\mathcal{R}$, ppt algorithms $\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{pSign}$ and dpt algorithms $\mathsf{Vrfy}, \mathsf{pVrfy}, \mathsf{Adapt}, \mathsf{Extract}$. Upon first activation, initialize this instance by running and storing $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$.

On $\texttt{PubKey}$? from anyone, return $(\texttt{PubKey}, \mathsf{pk})$.

On $(\texttt{Sign}, x)$ for party pid in session $sid = (pid, sid')$, do:
- $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, x)$
- **if** $\sigma \neq \bot$**: add** σ **to** $\mathsf{H}_{\mathsf{sign}}[x]$
- Return $(\texttt{Signature}, \sigma)$.

On $(\texttt{pSign}, x, Y)$ for party pid in session $sid = (pid, sid')$, do:
- $\widetilde{\sigma} \leftarrow \mathsf{pSign}(\mathsf{sk}, x, Y)$
- **if** $\neg\mathsf{corr} \wedge \exists x' \neq x : \widetilde{\sigma} \in \mathsf{H}_{\mathsf{pSign}}[x'] \cup \mathsf{H}_{\mathsf{pVerify}}[x']$**:** $\widetilde{\sigma} := \bot$ {*Message Bounding*
- **if** $\widetilde{\sigma} \neq \bot$**: add** $(\widetilde{\sigma}, Y)$ **to** $\mathsf{H}_{\mathsf{pSign}}[x]$
- Return $(\texttt{PreSignature}, \widetilde{\sigma})$

On $(\texttt{Adapt}, pk, \widetilde{\sigma}, y, Y)$ for party pid, do:
- $\sigma \leftarrow \mathsf{Adapt}(pk, \widetilde{\sigma}, y, Y)$
- **if** $pk = \mathsf{pk} \wedge (Y, y) \in \mathcal{R} \wedge \exists x : (\widetilde{\sigma}, Y) \in \mathsf{H}_{\mathsf{pVerify}}[x]$**:**
  - **add** $(\widetilde{\sigma}, Y, y, \sigma)$ **to** $\mathsf{H}_{\mathsf{adapt}}[x]$
- Return $(\texttt{Adapt}, \sigma)$

On $(\texttt{Extract}, pk, \widetilde{\sigma}, \sigma, Y)$ for party pid, do:
- $y := \mathsf{Extract}(pk, \widetilde{\sigma}, \sigma, Y)$
- **if** $(Y, y) \notin \mathcal{R} \wedge \exists (x, y') : (\widetilde{\sigma}, Y, y', \sigma) \in \mathsf{H}_{\mathsf{adapt}}[x]$**:** $y := y'$ {*Extract Correctness*
- Return $(\texttt{Extract}, y)$

On $(\texttt{pVerify}, pk, x, Y, \widetilde{\sigma})$ for party pid, do:
- $b_{\mathsf{pVerify}} := \mathsf{pVrfy}(pk, x, Y, \widetilde{\sigma})$
- **if** $pk = \mathsf{pk} \wedge (\widetilde{\sigma}, Y) \in \mathsf{H}_{\mathsf{pSign}}[x]$**:** $b_{\mathsf{pVerify}} := 1$ {*Pre-Sign Correctness*
- **else if** $pk = \mathsf{pk} \wedge \neg\mathsf{corr}$**:**
  - **if** $\exists x' \neq x : \widetilde{\sigma} \in \mathsf{H}_{\mathsf{pSign}}[x'] \cup \mathsf{H}_{\mathsf{pVerify}}[x']$**:** $b_{\mathsf{pVerify}} := 0$ {*Message Bounding*
- **if** $pk = \mathsf{pk} \wedge b_{\mathsf{pVerify}}$**: add** $(\widetilde{\sigma}, Y)$ **to** $\mathsf{H}_{\mathsf{pVerify}}[x]$
- Return $(\texttt{VerResult}, b_{\mathsf{pVerify}})$

On $(\texttt{Verify}, pk, x, \sigma)$ for party pid, do:
- $b_{\mathsf{vrfy}} := \mathsf{Vrfy}(pk, x, \sigma)$
- **if** $pk = \mathsf{pk} \wedge (x, \sigma) \in \mathsf{H}_{\mathsf{sign}}$**:** $b_{\mathsf{vrfy}} := 1$ {*Correctness*
- **else if** $pk = \mathsf{pk} \wedge \neg\mathsf{corr}$**:**
- **if** $pk = \mathsf{pk} \wedge b_{\mathsf{vrfy}}$**: add** σ **to** $\mathsf{H}_{\mathsf{verify}}[x]$
- Return $(\texttt{VerResult}, b_{\mathsf{vrfy}})$

As usual, the attacker may corrupt any party pid dynamically and gains full control over that party. In addition, if pid is the owner of the key pair modeled by this instance, i.e., the session ID of this instance is $sid = (pid, sid')$, then the attacker also learns sk; for notational convenience, we set $\mathsf{corr} = \mathsf{true}$ iff this is the case.

Fig. 19. Ideal adaptor signatures functionality $\mathcal{F}_{\text{asig}}^2$.

extracting, and verifying are public algorithms, so our reduction can simulate them. If the simulation comes to a point where the signer is corrupted, and the functionality would return the secret key, then the reduction attacker aborts the simulation. Such an abort does not change the distinguishability of $\mathcal{F}_{\text{asig}}^{(\text{ext},1)}$ and $\mathcal{F}_{\text{asig}}^{(\text{ext},2)}$, since we only enforce message-bound pre-signatures if the signer is not corrupted; hence, the functionalities only differ when the signer is not corrupted. So, if the signer is corrupted, both functionalities behave identically by definition, and nothing is visible. If no corruption happens, our reduction shows that the gap between the two worlds is at most negligible.

When the pre-signature oracle outputs a pre-signature $\widetilde{\sigma}$ on a message x, but the same pre-signature is already valid on x', or if the environment provides such a fresh, valid pre-signature to the pre-verification oracle, our reduction simply outputs the pre-signature and the matching messages. Since we just call efficient oracles polynomially often, our reduction is efficient. Since we assume that AS is correct, valid pre-signatures in our simulation are also valid w.r.t. the pre-verification algorithm.

Whenever the functionality $\mathcal{F}_{\text{asig}}^{(\text{ext},2)}$ differs from $\mathcal{F}_{\text{asig}}^{(\text{ext},1)}$, our reduction breaks the message-bound pre-signature property of AS. This holds since, by correctness, pre-signatures output by the pre-sign algorithm are valid. In addition, our reduction found a single pre-signature that is valid on two distinct messages (and the respective statements). As we assume AS to have message-bound pre-signatures, the gap between $\mathcal{F}_{\text{asig}}^{(\text{ext},2)}$ and $\mathcal{F}_{\text{asig}}^{(\text{ext},1)}$ is negligible in the security parameter.

Game $\mathcal{G}_3$. In this game, we enforce pre-signature adaptability. This means that, in this functionality, each signature obtained by adapting a pre-verifying pre-signature is guaranteed to verify. Besides this, the functionality $\mathcal{F}_{\text{asig}}^{(\text{ext},3)}$ has the same outputs as $\mathcal{F}_{\text{asig}}^{(\text{ext},2)}$. We depict $\mathcal{F}_{\text{asig}}^{(\text{ext},3)}$ in Figure 20.

Claim 7 *If AS, in addition to all the properties required by previous claims, satisfies pre-signature adaptability Definition 9, then the gap between $\mathcal{F}_{asig}^{(ext,3)}$ and $\mathcal{F}_{asig}^{(ext,2)}$ is negligible.*

Proof. The outputs of the functionalities $\mathcal{F}_{\text{asig}}^{(\text{ext},3)}$ and $\mathcal{F}_{\text{asig}}^{(\text{ext},2)}$ differ if the verification algorithm is called on a signature and a message x, such that the signature should be invalid by the verification algorithm, but the signature is part of the set $\mathsf{H}_{\mathsf{adapt}}[x]$. The signature can be part of $\mathsf{H}_{\mathsf{adapt}}[x]$, if there exists a pre-signature–statement pair, which is a part of $\mathsf{H}_{\mathsf{pVerify}}$, and the adapt oracle was called upon this pre-signature–statement pair and a valid witness for the mentioned statement. A pre-signature–statement pair is added to $\mathsf{H}_{\mathsf{pVerify}}$, if there exists a message x, such that either the pre-verification algorithm verifies on this triple, or the pre-signature was output by the pre-sign oracle, and hence pre-verifies by pre-sign correctness. Since pre-sign correctness is a perfect property, we know that in either case, the pre-verification algorithm on this pre-signature–statement pair, in addition to the message x, outputs 1.

This means, that $\mathcal{F}_{\text{asig}}^{(\text{ext},3)}$ differs from $\mathcal{F}_{\text{asig}}^{(\text{ext},2)}$, if a valid pre-signature w.r.t. some message x and some statement Y yields into an invalid adapted signature when adapted using a valid witness y for Y–effectively violating pre-signature adaptability. As we assume AS to have (perfect) pre-signature adaptability, there is no gap between $\mathcal{F}_{\text{asig}}^{(\text{ext},3)}$ and $\mathcal{F}_{\text{asig}}^{(\text{ext},2)}$.

At this point in the proof, not only the signatures stored in $\mathsf{H}_{\mathsf{sign}}[x]$ and $\mathsf{H}_{\mathsf{verify}}[x]$ verify w.r.t. some message x, but also all signatures stored in the set $\mathsf{H}_{\mathsf{adapt}}[x]$. This holds by pre-signature adaptability which is a perfect property.

Game $\mathcal{G}_4$. In this game, we enforce unique extractability as long as the signer remains uncorrupted. This is done by ensuring that the pre-sign oracle does not output a pre-signature that extracts with two valid signatures, and the sign oracle does not output a signature that extracts with a pre-signature that already has an extracting signature. We do the same in the Adapt Oracle. Furthermore, we ensure in the verification and pre-verification oracles that no (pre-)signature is valid if it allows extracting more than one (pre-)signature. We enforce this functionality by introducing the helper functions $\mathsf{BreakSigUnExt}$ and $\mathsf{BreakpSigUnExt}$ that, on input a message,(pre-)signature pair, output conflicting signatures which are considered valid. If this happens, the candidate (pre-)signatures are either considered invalid (in the verification case), the pre-sign and sign oracles output $\perp$ instead of such a conflicting (pre-)signature, and the adapt oracle outputs the conflicting

The functionality is parameterized by a polytime decidable relation $\mathcal{R}$, ppt algorithms $\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{pSign}$ and dpt algorithms $\mathsf{Vrfy}, \mathsf{pVrfy}, \mathsf{Adapt}, \mathsf{Extract}$. Upon first activation, initialize this instance by running and storing $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$.

On $\texttt{PubKey}$? from anyone, return $(\texttt{PubKey}, \mathsf{pk})$.

On $(\texttt{Sign}, x)$ for party pid in session $sid = (pid, sid')$, do:

$\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, x)$
if $\sigma \neq \bot$**: add** σ **to** $\mathsf{H}_{\mathsf{sign}}[x]$
Return $(\texttt{Signature}, \sigma)$.

On $(\texttt{pSign}, x, Y)$ for party pid in session $sid = (pid, sid')$, do:

$\widetilde{\sigma} \leftarrow \mathsf{pSign}(\mathsf{sk}, x, Y)$
if $\neg\mathsf{corr} \wedge \exists x' \neq x : \widetilde{\sigma} \in \mathsf{H}_{\mathsf{pSign}}[x'] \cup \mathsf{H}_{\mathsf{pVerify}}[x']$**:** $\widetilde{\sigma} := \bot$ {*Message Bounding*
if $\widetilde{\sigma} \neq \bot$**: add** $(\widetilde{\sigma}, Y)$ **to** $\mathsf{H}_{\mathsf{pSign}}[x]$
Return $(\texttt{PreSignature}, \widetilde{\sigma})$

On $(\texttt{Adapt}, pk, \widetilde{\sigma}, y, Y)$ for party pid, do:

$\sigma \leftarrow \mathsf{Adapt}(pk, \widetilde{\sigma}, y, Y)$
if $pk = \mathsf{pk} \wedge (Y, y) \in \mathcal{R} \wedge \exists x : (\widetilde{\sigma}, Y) \in \mathsf{H}_{\mathsf{pVerify}}[x]$**:**
  add $(\widetilde{\sigma}, Y, y, \sigma)$ **to** $\mathsf{H}_{\mathsf{adapt}}[x]$
Return $(\texttt{Adapt}, \sigma)$

On $(\texttt{Extract}, pk, \widetilde{\sigma}, \sigma, Y)$ for party pid, do:

$y := \mathsf{Extract}(pk, \widetilde{\sigma}, \sigma, Y)$
if $(Y, y) \notin \mathcal{R} \wedge \exists (x, y') : (\widetilde{\sigma}, Y, y', \sigma) \in \mathsf{H}_{\mathsf{adapt}}[x]$**:** $y := y'$ {*Extract Correctness*
Return $(\texttt{Extract}, y)$

On $(\texttt{pVerify}, pk, x, Y, \widetilde{\sigma})$ for party pid, do:

$b_{\mathsf{pVerify}} := \mathsf{pVrfy}(pk, x, Y, \widetilde{\sigma})$
if $pk = \mathsf{pk} \wedge (\widetilde{\sigma}, Y) \in \mathsf{H}_{\mathsf{pSign}}[x]$**:** $b_{\mathsf{pVerify}} := 1$ {*Pre-Sign Correctness*
else if $pk = \mathsf{pk} \wedge \neg\mathsf{corr}$**:**
  if $\exists x' \neq x : \widetilde{\sigma} \in \mathsf{H}_{\mathsf{pSign}}[x'] \cup \mathsf{H}_{\mathsf{pVerify}}[x']$**:** $b_{\mathsf{pVerify}} := 0$ {*Message Bounding*
if $pk = \mathsf{pk} \wedge b_{\mathsf{pVerify}}$**: add** $(\widetilde{\sigma}, Y)$ **to** $\mathsf{H}_{\mathsf{pVerify}}[x]$
Return $(\texttt{VerResult}, b_{\mathsf{pVerify}})$

On $(\texttt{Verify}, pk, x, \sigma)$ for party pid, do:

$b_{\mathsf{vrfy}} := \mathsf{Vrfy}(pk, x, \sigma)$
if $pk = \mathsf{pk} \wedge (x, \sigma) \in \mathsf{H}_{\mathsf{sign}}$**:** $b_{\mathsf{vrfy}} := 1$ {*Correctness*
else if $pk = \mathsf{pk} \wedge \exists (\widetilde{\sigma}, Y, y, \sigma) \in \mathsf{H}_{\mathsf{adapt}}[x]$**:** $b_{\mathsf{vrfy}} := 1$ {*Pre-Sig Adaptability*
else if $pk = \mathsf{pk} \wedge \neg\mathsf{corr}$**:**
if $pk = \mathsf{pk} \wedge b_{\mathsf{vrfy}}$**: add** σ **to** $\mathsf{H}_{\mathsf{verify}}[x]$
Return $(\texttt{VerResult}, b_{\mathsf{vrfy}})$

As usual, the attacker may corrupt any party pid dynamically and gains full control over that party. In addition, if pid is the owner of the key pair modeled by this instance, i.e., the session ID of this instance is $sid = (pid, sid')$, then the attacker also learns sk; for notational convenience, we set $\mathsf{corr} = \mathsf{true}$ iff this is the case.

Fig. 20. Ideal adaptor signatures functionality $\mathcal{F}_{\mathrm{asig}}^3$.

signature instead. Besides this, the functionality $\mathcal{F}_{\text{asig}}^{(\text{ext},4)}$ has the same outputs as $\mathcal{F}_{\text{asig}}^{(\text{ext},3)}$. We depict $\mathcal{F}_{\text{asig}}^{(\text{ext},4)}$ in Figure 21.

Claim 8 *If* AS*, in addition to all the properties required by previous claims, achieves unique extractability Definition 11, then the gap between* $\mathcal{F}_{asig}^{(ext,4)}$ *and* $\mathcal{F}_{asig}^{(ext,3)}$ *is negligible.*

Proof. The functionalities $\mathcal{F}_{\text{asig}}^{(\text{ext},3)}$ and $\mathcal{F}_{\text{asig}}^{(\text{ext},4)}$ differ, if the algorithms BreakSigUnExt and BreakpSigUnExt have non-bot outputs. To prove Claim 8, we show that the probability that the algorithms BreakSigUnExt and BreakpSigUnExt do not output $\perp$ when queried on valid (pre-)signatures is negligible. Since this is the only case where $\mathcal{F}_{\text{asig}}^{(\text{ext},4)}$ and $\mathcal{F}_{\text{asig}}^{(\text{ext},3)}$ differ, this shows the claim. To do this, we provide a reduction to the unique extractability of AS. As input, our reduction receives a verification key and has access to a signing and a pre-signing oracle. Using these oracles and the provided keys, our reduction instantiates our functionality $\mathcal{F}_{\text{asig}}^{(\text{ext},4)}$. Since we only ensure unique extractability if the signer is not corrupted, our reduction does not need to provide the corresponding signing key to the adversary. Eventually, either the BreakSigUnExt algorithm or the BreakpSigUnExt algorithm does not output $\perp$, but a signature σ. We now describe how our reduction finds values $(x, Y, \widetilde{\sigma}, \sigma, \sigma')$, such that both signatures verify on x, the pre-signature verifies on Y, x, and the pre-signature allows to extract with both signatures to a valid witness for Y.

We start with the algorithm BreakpSigUnExt, which has as input a valid pair $(x, \widetilde{\sigma}, Y)$. Having this input available, what is missing for our forgery against unique extractability are two valid signatures on x that both extract a valid witness with $(\widetilde{\sigma}, Y)$. When the algorithm BreakpSigUnExt is called, a set $\mathcal{Q}$ is filled with all verifying signatures (i.e. signatures from $\mathsf{H}_{\mathsf{sign}}[x]$ and $\mathsf{H}_{\mathsf{verify}}[x]$ that extract with $(\widetilde{\sigma}, Y)$. We capture all extracting signatures by running the Extract algorithm on the respective inputs. If BreakpSigUnExt outputs a signature which is non-$\perp$, this means that our list of extracting signatures contains at least two valid signatures $(\sigma_1 \neq \sigma_2)$ on x. Our reduction can return the pair $(x, \widetilde{\sigma}, Y, \sigma_1, \sigma_2)$ and successfully break unique extractability.

Now we consider the algorithm BreakSigUnExt, which has as input a valid pair (x, σ). This algorithm outputs non-$\perp$ if there exists a valid pre-signature–statement pair $(\widetilde{\sigma}, Y)$ which extracts with σ, but additionally extracts with a signature $\sigma' \neq \sigma$ that is already valid on x. To find valid pre-signature candidates, the algorithm first fills up a set $\mathcal{Q}$ containing all valid pre-signature–statement pairs $(\widetilde{\sigma}, Y)$ contained in $\mathsf{H}_{\mathsf{pSign}}[x]$ and $\mathsf{H}_{\mathsf{pVerify}}[x]$ that extract with σ. Those are valid pre-signatures on x w.r.t. some statement Y. Then, if there exists another signature $\sigma' \neq \sigma$ that was crafted via honest adapting (i.e. $\sigma' \in \mathsf{H}_{\mathsf{adapt}}[x]$), the algorithm returns such σ'. We know by extract correctness (Definition 8), that the extract algorithm yields a valid witness for a statement Y, when called upon a valid pre-signature on Y and the corresponding adapted pre-signature σ (which was obtained by adapting the pre-signature using a valid witness for Y). Therefore, each signature $\sigma' \in \mathsf{H}_{\mathsf{adapt}}[x]$ also extracts a valid witness with the pre-signature in question. Alternatively, if there exists a valid signature $(\sigma' \neq \sigma)$ that allows extracting with any pre-signature in $\mathcal{Q}$ using the Extract algorithm, the helper algorithm also outputs σ'. To craft a valid forgery, our reduction can return the pair $(x, \widetilde{\sigma}, Y, \sigma, \sigma')$, where σ' is the found signature, and $\widetilde{\sigma}$ is the matching pre-signature from $\mathcal{Q}$.

We have shown how to turn each non-$\perp$ output of the helper functions BreakSigUnExt and BreakpSigUnExt into a valid forgery of unique extractability. In addition, our reduction is efficient, since it internally runs a ppt adversary, and simulates oracles using provided oracles from the unique extractability security game. Hence, none of these algorithms can output a non-$\perp$ message during a run of the functionality if the AS satisfies unique extractability with non-negligible probability. Since the functionalities only have a gap if one of the helper algorithms outputs a non-bot message, the gap between $\mathcal{F}_{\text{asig}}^{(\text{ext},4)}$ and $\mathcal{F}_{\text{asig}}^{(\text{ext},3)}$ is negligible.

Game $\mathcal{G}_5$. In this game, we enforce unforgeability for pre-signatures for as long as the signer is uncorrupted. This means, in this functionality, a pre-signature can just be valid if it was either pre-signed before or if the message on which this pre-signature is pre-verified was signed before. Otherwise, the pre-verification algorithm outputs 0 in $\mathcal{F}_{\text{asig}}^{(\text{ext},5)}$. Besides this, the functionality $\mathcal{F}_{\text{asig}}^{(\text{ext},5)}$ has the same outputs as $\mathcal{F}_{\text{asig}}^{(\text{ext},4)}$. We depict $\mathcal{F}_{\text{asig}}^{(\text{ext},5)}$ in Figure 22.

The functionality is parameterized by a polytime decidable relation $\mathcal{R}$, ppt algorithms $\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{pSign}$ and dpt algorithms $\mathsf{Vrfy}, \mathsf{pVrfy}, \mathsf{Adapt}, \mathsf{Extract}$. Upon first activation, initialize this instance by running and storing $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$.

On $\mathtt{PubKey}$? from anyone, return $(\mathtt{PubKey}, \mathsf{pk})$.

On $(\mathtt{Sign}, x)$ for party pid in session $sid = (pid, sid')$, do:
- $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, x)$
- **if** $\mathsf{BreakSigUnExt}(x, \sigma) \neq \bot$: $\sigma := \bot$ {*Unique Extractability*
- **if** $\sigma \neq \bot$: **add** σ **to** $\mathsf{H}_{\mathsf{sign}}[x]$
- Return $(\mathtt{Signature}, \sigma)$.

On $(\mathtt{pSign}, x, Y)$ for party pid in session $sid = (pid, sid')$, do:
- $\widetilde{\sigma} \leftarrow \mathsf{pSign}(\mathsf{sk}, x, Y)$
- **if** $\neg\mathsf{corr} \wedge \exists x' \neq x : \widetilde{\sigma} \in \mathsf{H}_{\mathsf{pSign}}[x'] \cup \mathsf{H}_{\mathsf{pVerify}}[x']$: $\widetilde{\sigma} := \bot$ {*Message Bounding*
- **if** $\mathsf{BreakpSigUnExt}(x, \widetilde{\sigma}, Y) \neq \bot$: $\widetilde{\sigma} := \bot$ {*Unique Extractability*
- **if** $\widetilde{\sigma} \neq \bot$: **add** $(\widetilde{\sigma}, Y)$ **to** $\mathsf{H}_{\mathsf{pSign}}[x]$
- Return $(\mathtt{PreSignature}, \widetilde{\sigma})$

On $(\mathtt{Adapt}, pk, \widetilde{\sigma}, y, Y)$ for party pid, do:
- $\sigma \leftarrow \mathsf{Adapt}(pk, \widetilde{\sigma}, y, Y)$
- **if** $pk = \mathsf{pk} \wedge (Y, y) \in \mathcal{R} \wedge \exists x : (\widetilde{\sigma}, Y) \in \mathsf{H}_{\mathsf{pVerify}}[x]$:
  - **if** $\sigma' := \mathsf{BreakSigUnExt}(x, \sigma) \wedge \sigma' \neq \bot$: $\sigma \leftarrow \sigma'$ {*Unique Extractability*
  - **add** $(\widetilde{\sigma}, Y, y, \sigma)$ **to** $\mathsf{H}_{\mathsf{adapt}}[x]$
- Return $(\mathtt{Adapt}, \sigma)$

On $(\mathtt{Extract}, pk, \widetilde{\sigma}, \sigma, Y)$ for party pid, do:
- $y := \mathsf{Extract}(pk, \widetilde{\sigma}, \sigma, Y)$
- **if** $(Y, y) \notin \mathcal{R} \wedge \exists (x, y') : (\widetilde{\sigma}, Y, y', \sigma) \in \mathsf{H}_{\mathsf{adapt}}[x]$: $y := y'$ {*Extract Correctness*
- Return $(\mathtt{Extract}, y)$

On $(\mathtt{pVerify}, pk, x, Y, \widetilde{\sigma})$ for party pid, do:
- $b_{\mathsf{pVerify}} := \mathsf{pVrfy}(pk, x, Y, \widetilde{\sigma})$
- **if** $pk = \mathsf{pk} \wedge (\widetilde{\sigma}, Y) \in \mathsf{H}_{\mathsf{pSign}}[x]$: $b_{\mathsf{pVerify}} := 1$ {*Pre-Sign Correctness*
- **else if** $pk = \mathsf{pk} \wedge \neg\mathsf{corr}$:
  - **if** $\exists x' \neq x : \widetilde{\sigma} \in \mathsf{H}_{\mathsf{pSign}}[x'] \cup \mathsf{H}_{\mathsf{pVerify}}[x']$: $b_{\mathsf{pVerify}} := 0$ {*Message Bounding*
    - **if** $\mathsf{BreakpSigUnExt}(x, \widetilde{\sigma}) \neq \bot$: $b_{\mathsf{vrfy}} := 0$ {*Unique Extractability*
- **if** $pk = \mathsf{pk} \wedge b_{\mathsf{pVerify}}$: **add** $(\widetilde{\sigma}, Y)$ **to** $\mathsf{H}_{\mathsf{pVerify}}[x]$
- Return $(\mathtt{VerResult}, b_{\mathsf{pVerify}})$

On $(\mathtt{Verify}, pk, x, \sigma)$ for party pid, do:
- $b_{\mathsf{vrfy}} := \mathsf{Vrfy}(pk, x, \sigma)$
- **if** $pk = \mathsf{pk} \wedge (x, \sigma) \in \mathsf{H}_{\mathsf{sign}}$: $b_{\mathsf{vrfy}} := 1$ {*Correctness*
- **else if** $pk = \mathsf{pk} \wedge \exists (\widetilde{\sigma}, Y, y, \sigma) \in \mathsf{H}_{\mathsf{adapt}}[x]$: $b_{\mathsf{vrfy}} := 1$ {*Pre-Sig Adaptability*
- **else if** $pk = \mathsf{pk} \wedge \neg\mathsf{corr}$:
  - **if** $\mathsf{BreakSigUnExt}(x, \sigma) \neq \bot$: $b_{\mathsf{vrfy}} := 0$ {*Unique Extractability*
- **if** $pk = \mathsf{pk} \wedge b_{\mathsf{vrfy}}$: **add** σ **to** $\mathsf{H}_{\mathsf{verify}}[x]$
- Return $(\mathtt{VerResult}, b_{\mathsf{vrfy}})$

As usual, the attacker may corrupt any party pid dynamically and gains full control over that party. In addition, if pid is the owner of the key pair modeled by this instance, i.e., the session ID of this instance is $sid = (pid, sid')$, then the attacker also learns sk; for notational convenience, we set $\mathsf{corr} = \mathsf{true}$ iff this is the case.

Fig. 21. Ideal adaptor signatures functionality $\mathcal{F}_{\mathrm{asig}}^4$.

The functionality is parameterized by a polytime decidable relation $\mathcal{R}$, ppt algorithms $\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{pSign}$ and dpt algorithms $\mathsf{Vrfy}, \mathsf{pVrfy}, \mathsf{Adapt}, \mathsf{Extract}$. Upon first activation, initialize this instance by running and storing $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$.

On $\texttt{PubKey}$? from anyone, return $(\texttt{PubKey}, \mathsf{pk})$.

On $(\texttt{Sign}, x)$ for party pid in session $sid = (pid, sid')$, do:
- $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, x)$
- **if** $\mathsf{BreakSigUnExt}(x, \sigma) \neq \bot$: $\sigma := \bot$ {*Unique Extractability*
- **if** $\sigma \neq \bot$: **add** σ **to** $\mathsf{H}_{\mathsf{sign}}[x]$
- Return $(\texttt{Signature}, \sigma)$.

On $(\texttt{pSign}, x, Y)$ for party pid in session $sid = (pid, sid')$, do:
- $\widetilde{\sigma} \leftarrow \mathsf{pSign}(\mathsf{sk}, x, Y)$
- **if** $\neg\mathsf{corr} \wedge \exists x' \neq x : \widetilde{\sigma} \in \mathsf{H}_{\mathsf{pSign}}[x'] \cup \mathsf{H}_{\mathsf{pVerify}}[x']$: $\widetilde{\sigma} := \bot$ {*Message Bounding*
- **if** $\mathsf{BreakpSigUnExt}(x, \widetilde{\sigma}, Y) \neq \bot$: $\widetilde{\sigma} := \bot$ {*Unique Extractability*
- **if** $\widetilde{\sigma} \neq \bot$: **add** $(\widetilde{\sigma}, Y)$ **to** $\mathsf{H}_{\mathsf{pSign}}[x]$
- Return $(\texttt{PreSignature}, \widetilde{\sigma})$

On $(\texttt{Adapt}, pk, \widetilde{\sigma}, y, Y)$ for party pid, do:
- $\sigma \leftarrow \mathsf{Adapt}(pk, \widetilde{\sigma}, y, Y)$
- **if** $pk = \mathsf{pk} \wedge (Y, y) \in \mathcal{R} \wedge \exists x : (\widetilde{\sigma}, Y) \in \mathsf{H}_{\mathsf{pVerify}}[x]$:
  - **if** $\sigma' := \mathsf{BreakSigUnExt}(x, \sigma) \wedge \sigma' \neq \bot$: $\sigma \leftarrow \sigma'$ {*Unique Extractability*
  - **add** $(\widetilde{\sigma}, Y, y, \sigma)$ **to** $\mathsf{H}_{\mathsf{adapt}}[x]$
- Return $(\texttt{Adapt}, \sigma)$

On $(\texttt{Extract}, pk, \widetilde{\sigma}, \sigma, Y)$ for party pid, do:
- $y := \mathsf{Extract}(pk, \widetilde{\sigma}, \sigma, Y)$
- **if** $(Y, y) \notin \mathcal{R} \wedge \exists (x, y') : (\widetilde{\sigma}, Y, y', \sigma) \in \mathsf{H}_{\mathsf{adapt}}[x]$: $y := y'$ {*Extract Correctness*
- Return $(\texttt{Extract}, y)$

On $(\texttt{pVerify}, pk, x, Y, \widetilde{\sigma})$ for party pid, do:
- $b_{\mathsf{pVerify}} := \mathsf{pVrfy}(pk, x, Y, \widetilde{\sigma})$
- **if** $pk = \mathsf{pk} \wedge (\widetilde{\sigma}, Y) \in \mathsf{H}_{\mathsf{pSign}}[x]$: $b_{\mathsf{pVerify}} := 1$ {*Pre-Sign Correctness*
- **else if** $pk = \mathsf{pk} \wedge \neg\mathsf{corr}$:
  - **if** $\exists x' \neq x : \widetilde{\sigma} \in \mathsf{H}_{\mathsf{pSign}}[x'] \cup \mathsf{H}_{\mathsf{pVerify}}[x']$: $b_{\mathsf{pVerify}} := 0$ {*Message Bounding*
    - **if** $\mathsf{BreakpSigUnExt}(x, \widetilde{\sigma}) \neq \bot$: $b_{\mathsf{vrfy}} := 0$ {*Unique Extractability*
    - **if** $\mathsf{H}_{\mathsf{sign}}[x] = \emptyset$: $b_{\mathsf{pVerify}} := 0$ {*Pre-Sig Unforgeability*
- **if** $pk = \mathsf{pk} \wedge b_{\mathsf{pVerify}}$: **add** $(\widetilde{\sigma}, Y)$ **to** $\mathsf{H}_{\mathsf{pVerify}}[x]$
- Return $(\texttt{VerResult}, b_{\mathsf{pVerify}})$

On $(\texttt{Verify}, pk, x, \sigma)$ for party pid, do:
- $b_{\mathsf{vrfy}} := \mathsf{Vrfy}(pk, x, \sigma)$
- **if** $pk = \mathsf{pk} \wedge (x, \sigma) \in \mathsf{H}_{\mathsf{sign}}$: $b_{\mathsf{vrfy}} := 1$ {*Correctness*
- **else if** $pk = \mathsf{pk} \wedge \exists (\widetilde{\sigma}, Y, y, \sigma) \in \mathsf{H}_{\mathsf{adapt}}[x]$: $b_{\mathsf{vrfy}} := 1$ {*Pre-Sig Adaptability*
- **else if** $pk = \mathsf{pk} \wedge \neg\mathsf{corr}$:
  - **if** $\mathsf{BreakSigUnExt}(x, \sigma) \neq \bot$: $b_{\mathsf{vrfy}} := 0$ {*Unique Extractability*
- **if** $pk = \mathsf{pk} \wedge b_{\mathsf{vrfy}}$: **add** σ **to** $\mathsf{H}_{\mathsf{verify}}[x]$
- Return $(\texttt{VerResult}, b_{\mathsf{vrfy}})$

As usual, the attacker may corrupt any party pid dynamically and gains full control over that party. In addition, if pid is the owner of the key pair modeled by this instance, i.e., the session ID of this instance is $sid = (pid, sid')$, then the attacker also learns sk; for notational convenience, we set $\mathsf{corr} = \mathsf{true}$ iff this is the case.

Fig. 22. Ideal adaptor signatures functionality $\mathcal{F}_{\text{asig}}^5$.

Claim 9 *If* AS*, in addition to all the properties required by previous claims, has unforgeable pre-signatures (Definition 2), then the gap between* $\mathcal{F}_{asig}^{(ext,5)}$ *and* $\mathcal{F}_{asig}^{(ext,4)}$ *is negligible.*

Proof. The outputs of the functionalities $\mathcal{F}_{\text{asig}}^{(\text{ext},5)}$ and $\mathcal{F}_{\text{asig}}^{(\text{ext},4)}$ differ only if the signer is uncorrupted, and the pre-verification algorithm is called on a pre-signature and a message x, such that the pre-signature should be valid by the pre-verification algorithm, but the message x was never signed before, and the message-statement pair in question was not pre-signed before. In such cases, $\mathcal{F}_{\text{asig}}^{(\text{ext},5)}$ does not qualify the pre-signature to be valid, while $\mathcal{F}_{\text{asig}}^{(\text{ext},4)}$ would. We now show that the probability of such cases occurring is negligible in the security parameter. To this end, we provide a reduction that transforms any environment causing this deviation into an adversary that breaks the pre-signature unforgeability of AS. As input, our reduction receives a public key and has access to a signing and a pre-signing oracle that outputs valid (pre-)signatures w.r.t. the provided public key. To simulate the functionality in the environment, our reduction uses this key and the provided oracles to answer signing and pre-signing queries. Adapting, extracting, and verifying are public algorithms, so our reduction can simulate them while applying the rules established in $\mathcal{F}_{\text{asig}}^{(\text{ext},4)}$. Since we only enforce unforgeable pre-signatures for uncorrupted signers, the reduction does not need to provide the signing key to the adversary. When the pre-verification oracle is called on a pre-signature–statement pair such that the pre-verification algorithm outputs 1, but the corresponding message was never signed nor pre-signed w.r.t. this statement, our reduction outputs this pre-signature as forgery. This forgery is valid, since it was not output by the pre-sign oracle, and the message x was never signed before. Since we just call efficient oracles polynomially often, our reduction is efficient. Whenever the functionality $\mathcal{F}_{\text{asig}}^{(\text{ext},5)}$ differs from $\mathcal{F}_{\text{asig}}^{(\text{ext},4)}$, our reduction breaks the pre-signature unforgeability of AS. As we assume AS to have unforgeable pre-signatures, the gap between $\mathcal{F}_{\text{asig}}^{(\text{ext},5)}$ and $\mathcal{F}_{\text{asig}}^{(\text{ext},4)}$ is negligible in the security parameter.

Game $\mathcal{G}_6$**.** In this game, we enforce extractability for an uncorrupted signer. This means, in this functionality, each signature that is valid either was previously signed or allows extracting a valid witness together with a pre-signature previously output by the pre-signing oracle. Besides this, the functionality $\mathcal{F}_{\text{asig}}^{(\text{ext},6)}$ has the same outputs as $\mathcal{F}_{\text{asig}}^{(\text{ext},5)}$. Furthermore, this functionality is also our final functionality. We depict $\mathcal{F}_{\text{asig}}^{(\text{ext},6)}$ in Figure 23.

Claim 10 *If* AS*, in addition to all the properties required by previous claims, satisfies extractability (Definition 10), then the gap between* $\mathcal{F}_{asig}^{(ext,6)}$ *and* $\mathcal{F}_{asig}^{(ext,5)}$ *is negligible.*

Proof. The outputs of the functionalities $\mathcal{F}_{\text{asig}}^{(\text{ext},6)}$ and $\mathcal{F}_{\text{asig}}^{(\text{ext},5)}$ differ only if the signer is uncorrupted, and the verification algorithm is called on a signature and a message x, such that the signature should be valid by the verification algorithm, but the message was never previously signed, i.e. $\mathsf{H}_{\mathsf{sign}}[x] = \emptyset$, and there exists no pre-signature–statement pair $(\widetilde{\sigma}, Y) \in \mathsf{H}_{\mathsf{pSign}}[x]$ such that the extract algorithm does not output a valid witness for Y when called using the candidate signature and $\widetilde{\sigma}$. In such cases, $\mathcal{F}_{\text{asig}}^{(\text{ext},6)}$ does not qualify the signature to be valid, while $\mathcal{F}_{\text{asig}}^{(\text{ext},5)}$ would do so. We now show that the probability of such cases occurring is negligible in the security parameter. To this end, we provide a reduction that transforms any environment causing this deviation into an adversary that breaks the extractability of AS. As input, our reduction receives a public key and has access to a signing and a pre-signing oracle that outputs valid (pre-)signatures w.r.t. the provided public key. To simulate the functionality in the environment, our reduction uses this key and the provided oracles to answer signing and pre-signing queries. Adapting, extracting, and verifying are public algorithms, so our reduction can simulate them while applying the rules established in $\mathcal{F}_{\text{asig}}^{(\text{ext},5)}$.Since we enforce extractability only for honest signers, the reduction does not need to provide a matching signing key to the adversary. When the environment provokes a gap between $\mathcal{F}_{\text{asig}}^{(\text{ext},5)}$ and $\mathcal{F}_{\text{asig}}^{(\text{ext},6)}$, our reduction has available a signature–message pair such that the verification algorithm verifies, and there exists no pre-signature that qualifies for extracting a valid witness using the $\mathsf{Extract}$ algorithm, and the message was never signed before.

Since we just call efficient oracles and compute efficient algorithms polynomially often, our reduction is efficient. Whenever the functionality $\mathcal{F}_{\text{asig}}^{(\text{ext},6)}$ differs from $\mathcal{F}_{\text{asig}}^{(\text{ext},5)}$, our reduction breaks the extractability of

The functionality is parameterized by a polytime decidable relation $\mathcal{R}$, ppt algorithms $\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{pSign}$ and dpt algorithms $\mathsf{Vrfy}, \mathsf{pVrfy}, \mathsf{Adapt}, \mathsf{Extract}$. Upon first activation, initialize this instance by running and storing $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$.

On $\texttt{PubKey}$? from anyone, return $(\texttt{PubKey}, \mathsf{pk})$.

On $(\texttt{Sign}, x)$ for party pid in session $sid = (pid, sid')$, do:
- $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, x)$
- **if** $\mathsf{BreakSigUnExt}(x, \sigma) \neq \bot$: $\sigma := \bot$ {*Unique Extractability*
- **if** $\sigma \neq \bot$: **add** σ **to** $\mathsf{H}_{\mathsf{sign}}[x]$
- Return $(\texttt{Signature}, \sigma)$.

On $(\texttt{pSign}, x, Y)$ for party pid in session $sid = (pid, sid')$, do:
- $\widetilde{\sigma} \leftarrow \mathsf{pSign}(\mathsf{sk}, x, Y)$
- **if** $\neg\mathsf{corr} \wedge \exists x' \neq x : \widetilde{\sigma} \in \mathsf{H}_{\mathsf{pSign}}[x'] \cup \mathsf{H}_{\mathsf{pVerify}}[x']$: $\widetilde{\sigma} := \bot$ {*Message Bounding*
- **if** $\mathsf{BreakpSigUnExt}(x, \widetilde{\sigma}, Y) \neq \bot$: $\widetilde{\sigma} := \bot$ {*Unique Extractability*
- **if** $\widetilde{\sigma} \neq \bot$: **add** $(\widetilde{\sigma}, Y)$ **to** $\mathsf{H}_{\mathsf{pSign}}[x]$
- Return $(\texttt{PreSignature}, \widetilde{\sigma})$

On $(\texttt{Adapt}, pk, \widetilde{\sigma}, y, Y)$ for party pid, do:
- $\sigma \leftarrow \mathsf{Adapt}(pk, \widetilde{\sigma}, y, Y)$
- **if** $pk = \mathsf{pk} \wedge (Y, y) \in \mathcal{R} \wedge \exists x : (\widetilde{\sigma}, Y) \in \mathsf{H}_{\mathsf{pVerify}}[x]$:
  - **if** $\sigma' := \mathsf{BreakSigUnExt}(x, \sigma) \wedge \sigma' \neq \bot$: $\sigma \leftarrow \sigma'$ {*Unique Extractability*
  - **add** $(\widetilde{\sigma}, Y, y, \sigma)$ **to** $\mathsf{H}_{\mathsf{adapt}}[x]$
- Return $(\texttt{Adapt}, \sigma)$

On $(\texttt{Extract}, pk, \widetilde{\sigma}, \sigma, Y)$ for party pid, do:
- $y := \mathsf{Extract}(pk, \widetilde{\sigma}, \sigma, Y)$
- **if** $(Y, y) \notin \mathcal{R} \wedge \exists (x, y') : (\widetilde{\sigma}, Y, y', \sigma) \in \mathsf{H}_{\mathsf{adapt}}[x]$: $y := y'$ {*Extract Correctness*
- Return $(\texttt{Extract}, y)$

On $(\texttt{pVerify}, pk, x, Y, \widetilde{\sigma})$ for party pid, do:
- $b_{\mathsf{pVerify}} := \mathsf{pVrfy}(pk, x, Y, \widetilde{\sigma})$
- **if** $pk = \mathsf{pk} \wedge (\widetilde{\sigma}, Y) \in \mathsf{H}_{\mathsf{pSign}}[x]$: $b_{\mathsf{pVerify}} := 1$ {*Pre-Sign Correctness*
- **else if** $pk = \mathsf{pk} \wedge \neg\mathsf{corr}$:
  - **if** $\exists x' \neq x : \widetilde{\sigma} \in \mathsf{H}_{\mathsf{pSign}}[x'] \cup \mathsf{H}_{\mathsf{pVerify}}[x']$: $b_{\mathsf{pVerify}} := 0$ {*Message Bounding*
    - **if** $\mathsf{BreakpSigUnExt}(x, \widetilde{\sigma}) \neq \bot$: $b_{\mathsf{vrfy}} := 0$ {*Unique Extractability*
    - **if** $\mathsf{H}_{\mathsf{sign}}[x] = \emptyset$: $b_{\mathsf{pVerify}} := 0$ {*Pre-Sig Unforgeability*
- **if** $pk = \mathsf{pk} \wedge b_{\mathsf{pVerify}}$: **add** $(\widetilde{\sigma}, Y)$ **to** $\mathsf{H}_{\mathsf{pVerify}}[x]$
- Return $(\texttt{VerResult}, b_{\mathsf{pVerify}})$

On $(\texttt{Verify}, pk, x, \sigma)$ for party pid, do:
- $b_{\mathsf{vrfy}} := \mathsf{Vrfy}(pk, x, \sigma)$
- $b_{\mathsf{ext}} := \exists (\widetilde{\sigma}, Y) \in \mathsf{H}_{\mathsf{pSign}}[x] : (Y, \mathsf{Extract}(pk, \widetilde{\sigma}, \sigma, Y)) \in \mathcal{R}$
- **if** $pk = \mathsf{pk} \wedge (x, \sigma) \in \mathsf{H}_{\mathsf{sign}}$: $b_{\mathsf{vrfy}} := 1$ {*Correctness*
- **else if** $pk = \mathsf{pk} \wedge \exists (\widetilde{\sigma}, Y, y, \sigma) \in \mathsf{H}_{\mathsf{adapt}}[x]$: $b_{\mathsf{vrfy}} := 1$ {*Pre-Sig Adaptability*
- **else if** $pk = \mathsf{pk} \wedge \neg\mathsf{corr}$:
  - **if** $\mathsf{BreakSigUnExt}(x, \sigma) \neq \bot$: $b_{\mathsf{vrfy}} := 0$ {*Unique Extractability*
  - **if** $\mathsf{H}_{\mathsf{sign}}[x] = \emptyset \wedge \neg b_{\mathsf{ext}}$: $b_{\mathsf{vrfy}} := 0$ {*Extractability*
- **if** $pk = \mathsf{pk} \wedge b_{\mathsf{vrfy}}$: **add** σ **to** $\mathsf{H}_{\mathsf{verify}}[x]$
- Return $(\texttt{VerResult}, b_{\mathsf{vrfy}})$

As usual, the attacker may corrupt any party pid dynamically and gains full control over that party. In addition, if pid is the owner of the key pair modeled by this instance, i.e., the session ID of this instance is $sid = (pid, sid')$, then the attacker also learns sk; for notational convenience, we set $\mathsf{corr} = \mathsf{true}$ iff this is the case.

Fig. 23. Ideal adaptor signatures functionality $\mathcal{F}_{\text{asig}}^{6}$.

AS. As we assume AS to have extractability, the gap between $\mathcal{F}_{\text{asig}}^{(\text{ext},6)}$ and $\mathcal{F}_{\text{asig}}^{(\text{ext},5)}$ is negligible in the security parameter.

E Realization Proofs

Proof (Proof of Lemma 1). We present a concrete attack that demonstrates how Schnorr adaptor signatures violate pre-signature unforgeability. The attack leverages a technique we refer to as *statement shifting*. Let $(\mathsf{pk}, \mathsf{sk})$ be a Schnorr key pair, and let x be a message. Let $Y = g^y$ be a statement from the DLog hard relation, and let $\widetilde{\sigma} = (R \cdot Y, s)$ be a valid pre-signature on x for the statement Y, obtained via the pre-signing oracle. Here, R is the nonce commitment, and s is the masked response value. The Schnorr adaptor signature scheme computes the challenge as $c = H(\mathsf{pk}, R \cdot Y, x)$ and verifies a pre-signature (R', s) by checking that

$$g^s \cdot Y = R' \cdot \mathsf{pk}^c.$$

Now consider a new statement $Y' = Y \cdot g = g^{y+1}$ and define a new pre-signature

$$\widetilde{\sigma}' = (R',\ s - 1).$$

Note that $\widetilde{\sigma}'$ reuses the same commitment $R' = R \cdot Y = R \cdot Y' \cdot g^{-1}$ and the same message x, and thus the challenge remains unchanged, i.e., $c' = H(\mathsf{pk}, R', x) = c$. We now verify that $\widetilde{\sigma}'$ is a valid pre-signature on x with respect to Y'. The verification equation becomes:

$$g^{s-1} \cdot Y' = g^{s-1+y+1} = g^s \cdot Y = \mathsf{pk}^c \quad \Leftrightarrow \quad g^s = R' \cdot \mathsf{pk}^c,$$

which holds by assumption since $\widetilde{\sigma}$ was valid. Hence, $\widetilde{\sigma}'$ is a valid pre-signature on the same message x but with respect to the shifted statement Y'. Since $\widetilde{\sigma}'$ was computed without querying the pre-signing oracle on x and Y', this constitutes a successful forgery in the sense of pre-signature unforgeability (Definition 2). Therefore, Schnorr adaptor signatures do not satisfy pre-signature unforgeability.

Proof (Proof of Theorem 3). To prove Theorem 3, we proceed in two parts. First, we show that correctness, extractability, unique extractability, pre-signature adaptability, and message-bound pre-signatures of the enhanced Schnorr adaptor signature scheme reduce to the corresponding security properties of the original Schnorr adaptor signature construction and the completeness of the proof system. Second, we prove that the enhanced scheme additionally satisfies pre-signature unforgeability, a property not achieved by the original construction. Note that the PoK scheme of Figure 5 is simulation-sound extractable [BPW12] and zero-knowledge in the random oracle model (c.f. Appendix A).

Correctness. For correctness, we show both pre-sign correctness and extract correctness. For pre-sign correctness, we have to show that honest pre-signing on any statement and any message leads to a valid pre-signature, if the pre-signing algorithm does not fail. If the bitstring of the statement is parseable as a group element, and the message is of length ℓ_M, then this follows from the correctness of the standard Schnorr adaptor signature scheme and the (perfect) completeness of the NIZK proof. If not, the underlying Schnorr pre-signing algorithm will fail, and correctness holds trivially.

To establish extract correctness, we must show that for any valid pre-signature (σ_1, σ_2) on a message x with respect to a statement Y, and any valid witness y for Y, the extract algorithm correctly recovers y when given the adapted signature obtained via adapt. In our construction, adapt and extract are inverse operations. Adapting yields $(\sigma_1, \sigma_2 + y)$, which is well-defined since σ_2 and y lie in the same cyclic group. Applying extract then computes $(\sigma_2 + y) - \sigma_2 = y$. Thus, the extracted witness is identical to the one used for adapting and is valid for Y.

Extractability. We show that if extractability is violated in the enhanced Schnorr adaptor signature scheme, then it can also be broken in the underlying Schnorr scheme. To this end, we construct a reduction that uses an adversary $\mathcal{A}$ against extractability in the enhanced scheme to break extractability in the standard scheme.

The reduction proceeds by simulating the extractability experiment for the enhanced scheme using access to the (pre-)signing oracles of the extractability experiment for the standard scheme. The reduction forwards the key provided by the extractability experiment to the adversary. Whenever $\mathcal{A}$ issues a signing or pre-signing query, we forward this query to the standard (pre-)signing oracle to obtain a valid pre-signature $\widetilde{\sigma}$. We then use the zero-knowledge simulator of the proof of knowledge system to generate a simulated proof of knowledge for $\widetilde{\sigma}$, resulting in a valid pre-signature under the enhanced scheme. Since the proof system is zero-knowledge, this simulation is indistinguishable from a real proof with overwhelming probability.

Eventually, $\mathcal{A}$ outputs a message–signature pair (x, σ) such that σ is a valid Schnorr signature and no pre-signature was queried that allows extraction of the witness. By construction, σ contains a valid signature in the standard Schnorr adaptor signature scheme. Moreover, if the forgery is valid in the enhanced scheme, then x was never signed, and no corresponding pre-signature that extracts was queried in the enhanced scheme. Since any signature–pre-signature pair that extracts in the standard scheme also extracts in the enhanced scheme, the forgery against the enhanced scheme constitutes a valid forgery against extractability for the standard scheme. Our reduction is efficient, as the simulator and oracles are efficient. Hence, extractability in the enhanced Schnorr scheme reduces to extractability in the standard scheme.

Unique extractability. We show that if unique extractability is violated in the enhanced Schnorr adaptor signature scheme, then it can also be broken in the underlying standard Schnorr adaptor signature scheme. To this end, we construct a reduction that uses an adversary $\mathcal{A}$ against unique extractability in the enhanced scheme to break unique extractability in the standard scheme.

The reduction begins by forwarding the public key provided by the unique extractability experiment for the standard scheme to $\mathcal{A}$. It then simulates the enhanced experiment using access to the (pre-)signing oracles of the standard scheme. Whenever $\mathcal{A}$ issues a signing or pre-signing query, we forward this query to the standard (pre-)signing oracle to obtain a valid pre-signature $\widetilde{\sigma}$. We then use the zero-knowledge simulator of the proof of knowledge system to generate a simulated proof of knowledge for $\widetilde{\sigma}$, producing a valid enhanced pre-signature. Since the proof system is zero-knowledge, this simulation is indistinguishable from a real proof with overwhelming probability.

Eventually, $\mathcal{A}$ outputs a tuple $(x, \widetilde{\sigma}, Y, \sigma_1, \sigma_2)$ such that all (pre-)signatures are valid, and both σ_1 and σ_2 allow extracting valid witnesses for Y using $\widetilde{\sigma}$. This constitutes a violation of unique extractability in the enhanced scheme. By construction, all elements of this tuple—x, $\widetilde{\sigma}$, and σ_i—are also valid in the standard scheme (when ignoring the PoK). Moreover, since the extraction relation remains unchanged between the enhanced and standard schemes, the same pair $(\widetilde{\sigma}, \sigma_i)$ will extract to the same witness in both schemes. Hence, this tuple also constitutes a valid forgery against unique extractability in the standard Schnorr adaptor signature scheme. The reduction is efficient, as the simulation uses only efficient oracle access and a zero-knowledge simulator. Therefore, unique extractability in the enhanced Schnorr scheme reduces to unique extractability in the standard scheme.

Pre-signature adaptability. The same proof carries over to adaptability, since every valid pre-signature in the enhanced scheme contains a valid pre-signature in the standard scheme. In addition, we need to consider that we enhanced adaptability with respect to arbitrary-length messages. This transition works with the same argument as for correctness. So, by pre-signature adaptability of the standard scheme, this can be adapted into a valid signature. Signatures valid under the standard scheme are also valid in the enhanced scheme, since we did not modify the verification algorithm. So, also in the enhanced scheme, this signature is valid, and pre-signature adaptability holds.

Message-bound pre-signatures. Standard Schnorr adaptor signatures have message-bound pre-signatures, since the random oracle is collision resistant: The verification equation for a pre-signature (R', s) on a statement Y has the form

$$Y \cdot g^s = \mathsf{pk}^c \cdot R'$$

with $c = \mathsf{H}_{\mathsf{Sign}}(\mathsf{pk}, x, R')$. For a distinct message $x' \neq x$, we have that $c' = \mathsf{H}_{\mathsf{Sign}}(\mathsf{pk}, x', R') \neq c$ (modulo group order and with overwhelming probability). If $\mathsf{pk} \neq g^0$, then pk is a generator by primality of the group order such that $c' \neq c$ implies $\mathsf{pk}^{c'} \neq \mathsf{pk}^c$ and hence the verification equation does not hold for x'. Since the public key is chosen uniformly random over the group $\mathbb{G}$, the probability of being the neutral element g^0 is negligible. Observe that each pre-signature in our extended Schnorr adaptor signature scheme also contains a standard Schnorr adaptor pre-signature. Furthermore, pre-verification of the extended scheme requires validity of the standard Schnorr adaptor signature. Hence, the same argument also shows message-boundedness of our extended scheme.

Pre-signature unforgeability. Finally, we prove that the enhanced Schnorr adaptor signature scheme satisfies pre-signature unforgeability. We do this by a reduction to the simulation-sound extractability of the PoK and a reduction to the extractability of the standard Schnorr adaptor signature scheme. For this proof, simulation-sound extractability is necessary, since to win pre-signature unforgeability, the adversary is not required to output a valid signature, but just a pre-signature. However, if the adversary outputs a valid pre-signature w.r.t. some statement Y, the reduction might not know a valid witness w.r.t. Y, and hence, cannot adapt such a forgery into a valid signature. However, such a valid signature forgery would be needed to break (adaptor signature) extractability. Since this is not a valid proof strategy, we instead consider a different approach: As above, we build a reduction to the (adaptor signature) extractability and simulate proofs for pre-signatures using the zero-knowledge simulator. When the adversary outputs a valid pre-signature as forgery, this pre-signature contains a proof of knowledge of randomness. Using the extractor of the PoK, we extract this randomness, and by the algebraic structure, this allows us to learn the signing key and break extractability.

We now start our reduction to the extractability of standard Schnorr signatures. As input, the reduction receives a public key pk and has access to pre-sign and sign oracles. Our reduction forwards the public key to $\mathcal{A}$ and simulates signing queries by forwarding them to the underlying signature oracle. Pre-signing queries are handled by obtaining a standard pre-signature and then simulating a PoK using the zero-knowledge simulator. By the simulation-sound extractability of the PoK, this simulation is perfect with overwhelming probability. Eventually, $\mathcal{A}$ outputs a fresh pre-signature on a message x that was never pre-signed. At this point, we try to run the extractor of the PoK system to obtain the underlying randomness r.[4] Now, we have two cases: Either the extractor succeeds in extracting the randomness. If this is the case, and given this random value r and the forged pre-signature, we can extract the signing key sk, since the pre-signature contains the value $\mathsf{sk} \cdot h + r$ for a known h. Knowing sk, our reduction can break extractability for the Schnorr scheme, as it can now compute signatures on fresh messages. In the second case, where the extractor does not succeed, we use the power of $\mathcal{A}$ to break the simulation-sound extractability of the PoK system. Contained in the forgery, the adversary has output PoK for which the witness cannot be extracted; we forward the PoK to the extractor experiment for the proof system. This violates the simulation-sound extractability of the PoK system by definition. In both cases, the reduction is efficient and succeeds with non-negligible probability if $\mathcal{A}$ does.

[4] We note that the extractor is allowed to and typically has to rewind the adversary. A reader familiar with UC, where simulators cannot use rewinding, might wonder why this is possible without contradicting our UC security proof, i.e., Theorem 2. The reason is that our UC simulator does not actually have to run this extractor and hence does not have to perform any rewinding. Instead, we use this technique only as part of a contradiction argument to show that certain runs that would break the UC simulation have a negligible chance of occurring.