Improved Concurrent-Secure Blind Schnorr Signatures

Pierpaolo Della Monica and Ivan Visconti

Department of Computer, Control & Management Engineering
Sapienza University of Rome, Rome, Italy
{dellamonica,visconti}@diag.uniroma1.it

Abstract. Since the work of Chaum in '82, the problem of designing secure blind signature protocols for existing signature schemes has been of great interest. In particular, when considering Schnorr signatures, nowadays used in Bitcoin, designing corresponding efficient and secure blind signature schemes is very challenging in light of the ROS attack (Eurocrypt'21), which affected all previous efficient constructions.
Currently, the main positive result about concurrent-secure blind Schnorr signatures is the one of Fuchsbauer and Wolf (Eurocrypt'24). Their construction is quite demanding, indeed it requires trusted parameters, non-interactive zero-knowledge arguments and CPA-secure public-key encryption. Moreover, it is proven secure under a game-based definition only, is limited to computational blindness and is vulnerable to harvest now "link" later quantum attacks. Nicely, their construction is also a predicate blind signature (PBS) scheme, allowing signers to have some partial control on the content of the blindly signed message.
In this work, we show neat improvements to the state-of-the-art presenting a new construction for concurrent-secure blind Schnorr signatures that relies on milder/reduced cryptographic assumptions, enjoys statistical blindness, replaces the problematic trusted setup with a non-programmable random oracle (NPRO), and satisfies also a one-sided simulation-based property providing deniability in PBSs.

©IACR 2026. This is the full version of an article that will be published in the proceedings of Crypto 2026.

1 Introduction

A blind signature scheme [Cha82], allows a user to get from a signer a signature on a message satisfying: 1) **blindness**: the signer can not link a message-signature pair to a specific interaction; 2) **one-more unforgeability**: a malicious user will not get more than ℓ signatures when executing the protocol ℓ times only.

Definitions and known results. It is natural to consider a signer as an online service providing signatures to several clients that concurrently run sessions of the protocol asynchronously over the Internet. When using black-box simulation-based security definitions Lindell in [Lin03] proved the impossibility of concurrent-secure blind signatures. Later on in [GGS15] Goyal et al. gave a positive result using non-black-box simulation. Other positive results were achieved for blind signature schemes with simulation-based security leveraging trusted parameters and random oracles (e.g., [Fis06,AO09]). The above positive results fail in providing a efficient blind signature schemes for currently deployed signature schemes such as Schnorr's scheme, and this has motivated the downgrade to game-based security that therefore are by far the most used in the scientific literature.

Indeed, blind versions of popular signature schemes, such as BLS signatures [BLS01,Bol03], BBS signatures [TZ23] and RSA full-domain hash signatures [Cha82,DJW23], have been proven secure under game-based definitions. In several cases the blind signature protocols require also additional (and in some cases quite stronger) assumptions (e.g., interactive one-more variants of the underlying assumptions) compared to the security of the underlying signature scheme. Only very recently, efficient blind signature schemes were proposed under more standard assumptions (e.g., avoiding pairings, one-more variants). Chairattana-Apirom et al. [CATZ24] base their protocol on the CDH assumption, although it provides a weaker variant of unforgeability and produces signatures whose size depends on the security parameter. Kastner et al. [KNR24] rely on the strong RSA and DDH assumptions, with the notable advantage of being round-optimal. The Chairattana-Apirom et al. approach was subsequently improved by [KRW24,BHKR25], which rely on the DDH assumption and reduce the signature size. This line of work culminates in the recent work of Klooß and Reichle [KR25], which, assuming only DDH, remarkably achieves very short signatures and efficient communication size with a 5-round protocol. Still, such constructions satisfy game-based definitions, and do not produce signatures that match the popular signature schemes used in the wild.

Blind Schnorr signatures. Schnorr signatures are increasingly gaining popularity and are currently used also in Bitcoin, Bitcoin Cash, Litecoin, and Polkadot. For more than 30 years researchers have investigated the existence of an efficient and blind Schnorr signature scheme, focusing on the simpler game-based notion since this is notoriously already very challenging to achieve. The first construction, due to Chaum and Pedersen [CP93], was considered secure in the concurrent setting in [Sch01]. In that work, Schnorr introduced the "ROS problem" and proved unforgeability under the assumption that the ROS problem is hard. Wagner [Wag02] showed that ROS was not as hard as conjectured, proposing a subexponential-time attack. Two decades later, Benhamouda et al. [BLL+21] developed a polynomial-time algorithm that solves the ROS problem when polynomially many signing sessions are initiated, breaking concurrent security.

Fuchsbauer and Wolf [FW24] have recently proposed a concurrent-secure blind Schnorr signature scheme defeating the ROS attack, but with several shortcomings such as relying on some trusted parameters and on the security of NIZK arguments and CPA-secure encryption, on top of what is currently assumed by deployed Schnorr signatures (i.e., unforgeability when a concrete cryptographic hash function is used). Since the construction proposed by [FW24] is the only practical blind Schnorr signature scheme, reducing its hardness and setup assumptions (i.e., the points of failure) is of paramount importance. Furthermore, since the scheme of [FW24] lacks statistical blindness, achieving this property would be very beneficial, possibly obtaining also long-term security against quantum adversaries who could employ (kind of) "harvest now, link later" strategies to compromise in the future the blindness of signatures issued in the pre-quantum era.

Note that a previous attempt to cope with the ROS attack and thus obtain a concurrently secure construction appears in [FPS20]. That is, Fuchsbauer et al. propose a blind Schnorr protocol that is secure assuming a variant of ROS that they call mROS (together with an analysis in the AGM and RO models, and under the one-more discrete logarithm assumption). Unfortunately, under this new assumption, their solution incurs a significant security loss. Consequently, achieving standard security levels becomes prohibitive in terms of communication complexity. We refer directly to the comparison in [FW24] for more details.

Predicate blind signatures. There are scenarios where the signer might want to control parts of the (to be signed) message or enforce specific guarantees. The above motivated "Partially" blind signatures, first introduced in [AF96]. In this notion, a message consists of a public part (visible to the signer) and a secret part (kept hidden). This notion has been extensively explored, as it enables real-world protocols that are not possible to achieve by leveraging standard blind signatures, see [AO00,CHYC05,TZ22,CKM+23,BZ23] [HLW23,KRW24,AYY25] and references therein.

Next, Fuchsbauer and Wolf introduced the definition of a "Predicate" blind signature (PBS) [FW24], a generalization of partially blind signatures. PBS schemes allow the signer and user to agree on a predicate for the message that can be signed. The signer is guaranteed that the message satisfies the predicate, while the user is guaranteed that the signer does not learn more than that. They proved that PBSs naturally generalize regular (i.e., the classical blind signatures introduced by Chaum) and partially blind signatures.

Notice that a PBS scheme for a naïve predicate that is always satisfied is also a regular blind signature scheme.

1.1 Our Contributions

Primary contribution: improved assumptions for blind Schnorr signatures. As main result of this work, we present a new construction of a concurrent-secure blind Schnorr signature scheme significantly outperforming in multiple dimensions the one of [FW24], which we denote as FWSch. Indeed, in contrast with [FW24], our construction, denoted with PBSch, a) satisfies statistical blindness; b) can be instantiated without a trusted setup, and c) does not require additional hardness assumptions to the security of Schnorr scheme used in the wild (i.e., unforgeable when instantiated with a concrete standardized cryptographic hash function). An explicit comparison is provided in Table 1. In order to highlight the differences among the two works, we do not show the assumption required by both schemes, namely the security of Schnorr's instantiation from [FW24, Assumption 1], and we explicitly indicate the additional assumptions required by each specific construction. Note that, according to Table 1, our construction yields blind Schnorr signatures whose security in the NPRO model relies only on the security of the Schnorr signature scheme in the standard model. We also highlight that, while (as already stated by the authors in [FW24]) instantiating the protocol of [FW24] with a RO rather than with a CRS is straightforward, relying instead solely on an observable RO (i.e., a NPRO) is challenging, since their construction depends on programming the CRS, and they explicitly argue that similar programming is required in the RO model. Interestingly, our scheme is based on assumptions that are not known to imply key exchange, while the work of [FW24] requires the existence of Cryptomania.

Finally, we stress that there are other blind signature schemes [KLLQ24] that belong to the domain of Schnorr-style constructions, achieving in some cases also post-quantum security. While those constructions, inheriting the limitations of previous constructions of blind Schnorr signatures, have been considered so far of limited relevance in practice, our results shed new light towards obtaining practical instantiations also for those schemes.

For more details on our construction, we refer the reader directly to the technical overview. Finally, we remark that in the table we show a column about deniability, which is an additional security guarantee for the case of PBSs, that is a secondary result of our work that we will discuss shortly.

Table 1. Comparison of our construction with that of [FW24]. The **3th column** indicates the number of pairs of messages sent between the parties, following the notation of [FW24]. The **4th column** lists additional assumptions.

Scheme	Model Assumption	Rounds	Complexity Assumptions	Blindness	Deniability (only for PBS)
FWSch [FW24]	programmable CRS	2	PKE + NIZK	Computational	✗
PBSch (**this work**)	NPRO	3	–	Statistical	✓

Secondary result: deniability in predicate blind signatures (PBSs). We identify a limitation of blindness property of PBSs: in a scenario where the signer cannot efficiently sample a message that satifies the predicate (e.g., when sampling requires a trapdoor known to the user only), during the protocol the malicious signer could obtain a transferable evidence that a sample has been successfully performed, and this is something that the signer could not do on her own. Note that in PBSs [FW24], every session can

have a specific predicate, that is, starting from a base message space, e.g., $\{0,1\}^*$, the predicate could be satisfied only by messages belonging to a specific sub-message space $\mathcal{M}^* \subseteq \{0,1\}^*$.

We now discuss a toy example in the blockchain domain where the above issue can be exploited in a real-world attack. Consider the case where a blockchain miner computes a valid new block and would like to receive a signature of it (with a Schnorr signature) by a validator (e.g., an entity certifying the correctness of a block) before public disclosure. In our example, the miner does not trust the validator since validators might be interested in mounting a front-running attack. To avoid this, the miner engages blind signing process, naively assuming that game-based blindness will protect him from a malicious signer. We observe that, when using the scheme of [FW24] instantiated for PBSs (here the predicate will check that the message corresponds to a valid new block), the signer obtains a witness (i.e., a NIZK argument using the trusted parameters) that a new valid block has been discovered and this can be used against the miner (e.g., informing others not to keep mining to find a block at that depth since a valid block has been found already). Essentially, if the signer cannot efficiently sample a message that satisfies the predicate, then even a PBS scheme that is secure under the standard game-based definition may leak information to the signer.

Given the above limits of blindness in PBSs we propose an one-sided simulation-based property requiring that the interaction between an honest user and a malicious signer should be indistinguishable from the output of an expected PPT simulator that does not have the message in input but has black-box access to the signer. We will refer to this new definition as deniability[1]. We show that while the construction of a PBS scheme of [FW24] can be instantiated to satisfy blindness, yet it fails to provide deniability. Instead, our construction outperforms the one of [FW24] also with respect to this security guarantee. Finally, we remark that one may consider more relaxed notions of deniability (e.g., allowing the simulator to observe the adversary's RO queries). We leave the study of such settings and their implications to future work.

In Sec. 5 we also discuss the relation between our deniability notion and the standard definition of blindness, showing that (under some assumptions) they are incomparable. Thus, we believe that a predicate blind signature scheme should satisfy both notions to maximize security in heterogeneous use cases.

1.2 Technical Overview

We start describing the construction of the concurrent-secure blind Schnorr signature scheme of [FW24]. Their construction builds upon the blind Schnorr signature scheme introduced by Chaum and Pedersen [CP93], transforming it into a concurrent-secure variant. The construction from [FW24] works as follows.

Starting point: the construction of [FW24]. Let $(q, \mathbb{G}, G)$ be some concrete group parameters and H_q be the concrete hash function for an instantiation of Schnorr signatures. Let (x, X) be the signer's key pair (i.e., $x \in \mathbb{Z}_q$ and $X = xG$), such that the signing key is defined as $\mathsf{sk} := ((q, \mathbb{G}, G, \mathsf{H}_q, \mathsf{crs}, \mathsf{ek}), x)$ and the verification key as $\mathsf{vk} := ((q, \mathbb{G}, G, \mathsf{H}_q, \mathsf{crs}, \mathsf{ek}), X)$. Here, crs and ek are respectively a CRS for a NIZK proof system and an encryption key for a PKE scheme, in the construction of [FW24] for a message space $\mathcal{M}$.

The user first samples two *blinding values* $(\alpha, \beta) \leftarrow\!\!\$\ \mathbb{Z}_q$ and along with the message $m \in \mathcal{M}$ to be signed, computes the encryption $C := \mathsf{PKE.Enc}(\mathsf{ek}, (m, \alpha, \beta); \rho)$ where ρ is the algorithm randomness. The ciphertext C is then sent to the signer. The signer samples $r \leftarrow\!\!\$\ \mathbb{Z}_q$ and computes $R := rG$, sending R to the user. The user computes $R' := R + \alpha G + \beta X$, which will be the first component of the blind signature, and next it computes $c := \mathsf{H}_q(R', X, m) + \beta$ along with a NIZK proof π for the instance $\mathsf{stm} := (X, R, c, C, \mathsf{ek})$ and witness $\mathsf{wtn} := (m, \alpha, \beta, \rho)$ proving that: $(c := \mathsf{H}_q(R + \alpha G + \beta X, X, m) + \beta) \wedge (m \in \mathcal{M}) \wedge (C := \mathsf{PKE.Enc}(\mathsf{ek}, (m, \alpha, \beta); \rho))$. The user then sends c and π to the signer. The signer verifies π and, if valid, replies with $s := (r + cx)$ (for simplicity, we do not write modular reduction explicitly when it is clear from the context). Finally, the user computes $s' := (s + \alpha)$, yielding the signature (R', s'), which satisfies the Schnorr verification equation: $s'G = R' + \mathsf{H}_q(R', X, m)X$.

[1] We adopt the term "deniability" from the zero-knowledge literature [Pas04], where it describes a similar simulation-based property.

Deniability. We notice that during the above interaction, the signer receives the NIZK proof π, which attests that a message was sampled correctly[2]. Since this proof is undeniable, the signer can exploit it, and this is an information she could not have generated on her own when she cannot efficiently sample messages satisfying the predicate. This subtle issue is a limitation of the standard game-based definition of blindness. Therefore, we propose an additional simulation-based definition where an efficient simulator that can not efficiently sample messages, that satisfies the predicate, can replace the user, producing an indistinguishable transcript. We call such property deniability. The scheme of [FW24] can be instantiated so that it is secure according to the blindness definition, but it is not according to deniability.

While the above shows that a scheme enjoying the standard game-based blindness might not enjoy deniability, we also notice that there can be blind signature schemes that generate signatures with sufficient min-entropy and satisfy deniability. In such schemes, one can extract randomness from this min-entropy and transmit it to the signer in the final protocol step without violating deniability, since this merely augments the signer's view with a random string. However, this modified blind signature protocol clearly fails to satisfy standard game-based blindness, as the signer could subsequently use the revealed randomness to identify which session produced a given message-signature pair. In the technical sections that follow, we formalize the distinction between these two notions and demonstrate how they capture fundamentally different requirements.

Improved blind Schnorr signatures. Starting from the blind signature scheme in [FW24], we propose a new blind signature scheme that also satisfies deniability. The scheme works according to the signing and verification keys $\mathsf{sk} := ((q, \mathbb{G}, G, \mathsf{H}_q, \mathsf{ck}), x, x')$ and $\mathsf{vk} := ((q, \mathbb{G}, G, \mathsf{H}_q, \mathsf{ck}), X, X')$ where (x', X') is an auxiliary (Schnorr) signing key pair[3], and ck is a key for a commitment scheme[4].

Our protocol proceeds similarly to [FW24] until the signer sends R to the user. The only difference is that we (generically) use a non-interactive straight-line extractable commitment [Pas04] Cmt, defined with the following algorithms $\mathsf{Cmt} = (\mathsf{Cmt.Setup}, \mathsf{Cmt.Com}, \mathsf{Cmt.Dec})$, instead of a PKE scheme for computing C. We emphasize that [FW24] explicitly states that they cannot generalize their construction replacing the encryption with this type of commitments (that might be instantiated through milder/different hardness assumptions). In contrast, we make several modifications to their construction, as detailed below, that allow us to use some special extractable commitments. The special requirement is about the need to prove that the extractable commitment has been computed correctly without opening it. Indeed, we cannot use the simple commitment of Pass [Pas04], because it would require to compute a proof about the correct output of a RO, but we will show other instantiations that avoid such shortcoming.

Having outlined the key differences, we now describe our construction in detail. Along with R, the signer additionally sends a uniformly sampled string ν_s. The user also samples a random string ν_u and sends it to the signer. The signer then sends to the user a Schnorr signature σ on the message (ν_s, ν_u). That is, the signer computes $\sigma \leftarrow \mathsf{Sch.Sign}(\mathsf{sk}'_{\mathsf{Sch}}, (\nu_s, \nu_u))$ where $\mathsf{sk}'_{\mathsf{Sch}} := (q, \mathbb{G}, G, \mathsf{H}_q, x')$.

The user first checks that σ is a valid signature for (ν_s, ν_u) with respect to $\mathsf{vk}'_{\mathsf{Sch}} := (q, \mathbb{G}, G, \mathsf{H}_q, X')$. Then the protocol continues as the one in [FW24] with the user sending $c := \mathsf{H}_q(R + \alpha G + \beta X, X, m) + \beta$ and a proof π that c is computed correctly with respect to the blinding values and the message, which are committed within C sent by the user in the first message.

The core difference lies in the proof π specifically in our protocol the proof is computed with a non-interactive WI (NIWI) argument system with statement $\mathsf{stm} := (X, X', R, c, C, \mathsf{ck}, S)$ and witness $\mathsf{wtn} := (m, \alpha, \beta, \rho, \sigma, \sigma', \nu_u, \nu'_u, \nu_s, \varrho)$ proving that:

$$\begin{aligned}(c := \mathsf{H}_q(R + \alpha G + \beta X, X, m) + \beta \wedge C := \mathsf{Cmt.Com}(\mathsf{ck}, (m, \alpha, \beta); \rho) \wedge m \in \mathcal{M}) \vee \\ (\nu_u \neq \nu'_u \wedge S = \mathsf{Cmt.Com}(\mathsf{ck}, (\sigma, \sigma', \nu_u, \nu'_u, \nu_s); \varrho) \wedge \\ \mathsf{Sch.Verify}(\mathsf{vk}'_{\mathsf{Sch}}, (\nu_s, \nu_u), \sigma) = 1 \wedge \mathsf{Sch.Verify}(\mathsf{vk}'_{\mathsf{Sch}}, (\nu_s, \nu'_u), \sigma') = 1)\end{aligned} \tag{1}$$

That is, π proves that either c is correctly computed with respect to the commitment C or that the user knows a commitment S of two valid signatures (σ, σ') on two different messages that share the same message prefix ν_s but have different suffixes ν_u and ν'_u. Since the (malicious) user does not know two

[2] For simplicity of exposition, we say that the NIZK attests that a message was sampled correctly. In reality, it guarantees that the message satisfies the predicate, or in other words, it is part of the sub-message space that is the space of all messages that satisfy the predicate.

[3] While x and X are the secret and public keys for Schnorr signatures, x' and X' are auxiliary keys to be used during the execution of the blind signature protocol.

[4] That is, for example if the commitment is instantiated with an encryption scheme, then ck is the corresponding public key; if it is a Pedersen commitment, then ck consists of the group description and the two generators.

valid signatures that share the same prefix, the argument system guarantees that c is correctly computed and thus S is computed by the (malicious) user as a commitment of a "garbage" message.

Core idea: constant-round concurrent statistical ZK with on-line extraction in the NPRO model. A major deviation between our construction and the one of [FW24] is the replacement of the NIZK sent by the user to the signer with a concurrent statistical ZK argument of knowledge with a straigh-line extractor that we construct in four messages (i.e., two rounds) relying on an NPRO and on the fact that multiple sessions are all played with the same verifier (i.e., the signer, identified by the public key). This allows us to remove the need for a programmable random oracle (or a programmable CRS), therefore improving both on the assumptions and obtaining deniability as a byproduct.

Notice that we want a constant-round blind signature protocol, but black-box concurrent ZK in constant rounds is impossible for non-trivial languages [CKPR01], and this is a complication that requires to accurately build this subprotocol.

The four messages of our ZK argument of knowledge are due to those three messages allowing the simulator to extract a trapdoor witness, followed by a NIWI argument of knowledge with straight-line extraction in the NPRO model, that can be constructed as example by applying the Fiat-Shamir transform to Blum's protocol for Graph Hamiltonicity, where commitments are statistically hiding and computed via queries to the NPRO[5].

We crucially notice that in the setting of blind signatures, we are already in a stronger model for which concurrent-secure constant-round statistical ZK arguments of knowledge are possible but without straight-line extraction. Specifically, we observe that every signer (i.e., the verifier of the ZK proof) has a fixed identity that remains the same across all sessions (namely, the public key). In the literature, this setting has been named the bare public-key model [CGGM00].

We leverage this setting to construct our interactive statistical ZK argument of knowledge, where we additionally use the NPRO to obtain straight-line extraction of the witness.

More in detail, in our construction, the signer (i.e., the verifier) has a fixed identity pk and as first message sends a random string ν_s while the prover (i.e., the user) sends a random string ν_u. In the third message, the signer/verifier sends a signature σ on the message (ν_s, ν_u) according to pk. This signature will be included in the statement that will be next proven through a NIWI by the user/prover. In particular, the statement will claim that either the computation of c (i.e., the evaluation of the cryptographic hash function used in Schnorr's signatures) has been performed correctly or the prover knows two signatures with respect to pk where the messages have the same prefix, as described in (1). The simulator will leverage an extraction strategy that works in the bare public-key model, and thus also in our setting. In particular, the simulator extracts a trapdoor (i.e., the signatures of two messages with the same prefix) from a single session and this can be used as a trapdoor witness in all sessions. This trapdoor is then used by the simulator in the NIWI. Noticing that the NIWI can be instantiated with statistical WI and straight-line extraction using the NPRO, we obtain the desired result.

Security of our scheme. We prove that our scheme satisfies strong one-more unforgeability, along with statistical blindness and statistical deniability. In a nutshell, in the proof of deniability (the same idea applies in the blindness proof), the simulator mimics an honest user, with two key differences. First, it commits to a "garbage" value (with the blinding parameters) instead of the actual message. Second, using its black-box access to the adversary, the simulator rewinds the adversary to extract a trapdoor to be used as witness for generating the proof π, specifically, obtaining two valid signatures for messages that share a common prefix but differ in the suffix. The core idea of the simulation rewind strategy is that the simulator only needs to extract a tuple $(\sigma, \sigma', \nu_u, \nu'_u, \nu_s)$ such that both signatures σ and σ' are valid on messages (ν_s, ν_u) and (ν_s, ν'_u) respectively. That is, they share a common prefix ν_s. Once such a tuple is extracted in one session, it can be used for simulating all sessions. Indeed, we have carefully designed the above claim (1) to make sure that the extraction of one single trapdoor in one session is enough (and can be reused) to obtain a valid witness in all the sessions.

Although we provide a detailed proof and analysis of the simulator's running time and extraction probability under our rewinding strategy in the technical sections, we emphasize that this strategy is not novel, it is already carefully analyzed in the literature and has been used in multiple influential

[5] This technique guarantees straight-line extraction and can be applied here (unlike in the extractable commitment played in the very first round of our protocol) because there will not be a proof about these committed values (which could be analyzed only heuristically since a proof over the circuit of a random oracle makes no sense).

papers on concurrent/resettable zero-knowledge in the bare public-key model[6]; our simulator's behavior closely mirrors the one of Canetti et al. [CGGM00], that works in the even more demanding setting of resettable (rather than just concurrent as in our case) zero-knowledge in the bare public-key model with polynomially many different verifiers (rather than just one verifier, the signer in our case). In the very same way, in our case, the simulator performs a single successful extraction and reuses the extracted information in multiple concurrent sessions.

As mentioned earlier, we also explore how to instantiate the non-interactive straight-line extractable commitment scheme. Specifically, inspired in part by techniques from [GKO+23,KRW24], we propose an instantiation in the NPRO model, based on Pedersen commitments, that ensures both circuit representability and straight-line extractability. That is, we extend the standard Pedersen commitment with a straight-line extractable WI proof of correct opening in the NPRO model. This is achieved by transforming the standard Σ-protocol for opening a Pedersen commitment into a straight-line extractable WI proof, following the classical approaches of [Pas04,Fis05]; we defer more details on this to the main body. Notably, the statistical hiding of such commitment (together with further considerations on how the WI argument system can be instantiated) lets us upgrade blindness from *computational* to *statistical*, thus improving over [FW24], where blindness holds only against efficient signers. Note that, achieving statistical blindness provides long-term security against adversaries who could otherwise employ (kind of) "harvest now, decrypt later" strategies to compromise the blindness in the future, making our scheme relevant for post-quantum security considerations.

Comparison with [FW24]. Finally, we provide a detailed comparison with [FW24], analyzing both the security assumptions and the practical efficiency of our constructions (see Table 1). Regarding efficiency, following the approach in [FW24], we measure the arithmetic complexity[7] of the relation in (1) and compare it with the arithmetic complexity measured in [FW24]. According to [FW24], issuing such a proof is the most demanding operation that could, in principle, make the construction impractical. Note that although our construction requires computing also a straight-line extractable argument of knowledge to build the straight-line extractable commitment, by using the transform of [Fis05], such computation is very efficient (comparable to computing a standard Schnorr signature) according to benchmarks conducted in [CL24]. We provide a succinct comparison in Table 2 and defer a more in-depth analysis to the technical sections. Namely, in Table 2, as in [FW24], we consider two types of scenarios: **(A)** The group and hash functions used for the Schnorr signature parameters $(q, \mathbb{G}, G, \mathsf{H}_q)$, with respect to the verification key X, can be chosen based on the specifics of the Niwi. **(B)** The protocol is intended to extend an existing implementation of Schnorr signatures for given parameters $(q, \mathbb{G}, G, \mathsf{H}_q)$, and such a standard (e.g., as those used by Bitcoin) verification key X[8].

Table 2. Benchmark of the arithmetic complexity for the relation $\mathsf{R}_{\mathsf{FWSch}}$ used in the PBS of [FW24], namely FWSch, and for the relation $\mathsf{R}_{\mathsf{PBSch}}$ used in the PBSch scheme across different scenarios. The first and second rows describe the Schnorr parameters used: the elliptic curve (BJB for Baby JubJub and secp256k1 for Bitcoin) and the hash function (Poseidon [GKR+21] and SHA-256). Blindness type refers to the degree of blindness applied (full, partial, or predicate-based), and Message size indicates the size of the messages in bytes (B) or bits (b). Hardwiring denotes whether verification and commitment keys are hardwired maximal or minimal. Concerning the definition of a parameterized relation R (see App. A.5), maximal hardwiring means that both keys are in $\mathsf{par}_{\mathsf{R}}$ while minimal that are in the statement stm.

Scenario	**(A1)**	**(A2)**	**(A3)**	**(B1)**	**(B2)**	**(B3)**
Schnorr curve	BJB	BJB	BJB	secp256k1	secp256k1	BJB
Schnorr hash	Poseidon	Poseidon	Poseidon	SHA-256	SHA-256	SHA-256
Blindness type	full	full	partial	full	predicate	predicate
Message size	253 b	253 B	252 B	256 b	256 B	256 B
Hardwiring	max.	min.	min.	min.	min.	min.
$\mathsf{R}_{\mathsf{FWSch}}$ ([FW24])	4 226	8 830	8 404	1 564 556	1 716 794	219 740
$\mathsf{R}_{\mathsf{PBSch}}$ (**this work**)	7 857	15 997	15 559	1 571 750	1 724 071	227 008

[6] In the bare public-key model, each verifier is assumed to have deposited a public key in a publicly (and fixed) accessible file before sessions start. The same holds with the definition of blind signatures, where the signer is assumed to chosen the key that will then be used in multiple sessions.

[7] Sometimes referred to as the "number of constraints" when expressed in R1CS [BCR+19].

[8] For details on settings/optimizations for $\mathsf{R}_{\mathsf{FWSch}}$, refer to [FW24, Section 5].

In summary, we improve upon the blind Schnorr signature scheme of [FW24] along several dimensions. First, we improve the underlying construction by removing the need for a trusted setup. Our scheme achieves this by relying instead on the NPRO model. In contrast, [FW24] acknowledges that the requirement of a trusted setup is undesirable, both because the standard Schnorr signature scheme does not require it and because a possible failure of the trusted setup is a serious security concern. Their proposed mitigation introduces additional "knowledge-type" assumptions to provide resilience against such failures, but this comes at the cost of reduced security. Our construction avoids these drawbacks entirely. Second, while [FW24] achieves both blindness and one-more unforgeability against efficient adversaries, we upgrade blindness to statistical security (thus achieving resistance to "harvest now, link later" type of attacks). Our constructions align concurrent blind Schnorr signatures with other well-known state-of-the-art blind signature schemes [TZ22,CAHL+22,HLW23,CATZ24], where blindness is typically statistical. Third, we significantly reduce the underlying assumptions (that in turn are points of failure). Both our work and [FW24] rely on the security of the instantiated Schnorr signatures (i.e., assuming that the real-world implementation of is secure when instantiated with an hash function), to cope with the "ROS attack". However, while [FW24] additionally requires stronger primitives, specifically, PKE and NIZK, our construction avoids these entirely. As a result, our scheme relies only on the security of the "instantiated" Schnorr signature, in the NPRO model. This result is somewhat optimal, as the output of any blind Schnorr signature is a Schnorr signature, and thus its security inherently depends on the assumption that Schnorr is secure when instantiated in practice. Finally, we identify a limitation in the game-based definition of blindness that affects also the construction of [FW24]. We address this by introducing a stronger, one-sided simulation-based notion of security, which we refer to as deniability.

Our construction requires one more round (i.e., two messages) and following the same approach as [FW24], we have shown that the arithmetic complexity of our more complex relation is only slightly greater than the one of [FW24].

2 Preliminaries

Standard definitions, notations are provided in App. A. We adopt the same notation of Fuchsbauer and Wolf [FW24], which is very standard.

We also discuss in App. A.6 a possible instantiation of straight-line extractable commitment schemes secure in the NPRO, which we denote by $\mathsf{Cmt} = (\mathsf{Cmt.Setup}, \mathsf{Cmt.Com}, \mathsf{Cmt.Dec})$. In a nutshell, we use a straight-line extractable commitment based on Pedersen commitment together with a non-interactive straight-line extractable proof of opening (i.e., obtained via Fischlin's transformation of the Σ-protocol for the Pedersen commitment opening). Such extractable commitment follows the very same ideas in [GKO+23,KRW24].

Following Fuchsbauer and Wolf, we adopt the same assumption on Schnorr signatures, which we restate below for clarity. Note that in the assumption, the GrGen algorithm outputs a (believed hard) group description and a generator for such a group, the HGen algorithm outputs a description of an hash function, and the $\mathsf{Sch}[\mathsf{GrGen}, \mathsf{HGen}]$ scheme is the Schnorr signature parameterized over GrGen and HGen.

Assumption 1 ([FW24, Assumption 1]). *There exists a group generator* GrGen *and a hash function generator* HGen *such that the Schnorr signature scheme* Sch*, described in Fig. 8, is strongly unforgeable; in particular, for every* PPT *adversary* A*, the advantage* $\mathsf{Adv}^{\text{SUF-CMA}}_{\mathsf{Sch}[\mathsf{GrGen},\mathsf{HGen}]}(\mathsf{A}, \lambda)$ *is negligible in* λ*.*

2.1 (Predicate) Blind Signature Schemes

Here we provide the definition of (predicate) blind signatures (PBS) according to the security model presented in [FW24]. The concept of a PBS was introduced in [FW24], as a generalization of standard and partially blind signatures [AF96]. A PBS scheme is an interactive protocol that enables a signer to sign a message for another party, called the user, with a security guarantee that the signer cannot obtain any information about the signed message, except that it satisfies certain conditions (defined by a predicate) on which the user and signer agreed upfront.

A PBS scheme is parameterized by a family of polynomial time computable predicates, which are implemented by a polynomial time algorithm $\mathsf{P} : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}$, the predicate complier, that on input a predicate description $\varphi \in \{0,1\}^*$ and a message $m \in \{0,1\}^*$, outputs 1 if m satisfies φ, otherwise 0. Note that a blind signature scheme can be easily obtained from a PBS scheme by using a predicate compiler that for every message m always outputs 1 regardless of φ, or in other words, using an "empty" predicate description φ.

A PBS scheme $\mathsf{PBS}[\mathsf{P}]$ for P is defined by the following algorithms.

- $\mathsf{PBS.Setup}(1^\lambda) \to \mathsf{par}$ on input the security parameter, outputs public parameters par, which define a message space $\mathcal{M}_{\mathsf{PBS}}$.
- $\mathsf{PBS.KeyGen}(\mathsf{par}) \to (\mathsf{sk}, \mathsf{vk})$ on input the parameters par, outputs a signing/verification key pair $(\mathsf{sk}, \mathsf{vk})$, implicitly containing par: i.e., $\mathsf{par} :\subseteq \mathsf{vk}$ and $\mathsf{par} :\subseteq \mathsf{sk}$.
- $\langle \mathsf{PBS.Sign}(\mathsf{sk}, \varphi), \mathsf{PBS.User}(\mathsf{vk}, \varphi, m)\rangle \to (0/1, \sigma/\perp)$ is an *interactive protocol*, with shared input par (implicit in sk and vk) and a predicate φ, that is run between two PPT algorithms $\mathsf{PBS.Sign}$, the signer algorithm (or the signer) and $\mathsf{PBS.User}$, the user algorithm (or the user). The signer takes a signing key sk as private input, the user's private input is a verification key vk and a message m. The signer outputs 1 if the interaction completes successfully and 0 otherwise, while the user outputs a signature σ if the interaction completes successfully, and $\perp$ otherwise.
  Both $\mathsf{PBS.Sign}$ and $\mathsf{PBS.User}$ are composed by several sub algorithms that are executed depending on the input received during the interaction, namely $\mathsf{PBS.Sign} = (\mathsf{PBS.Sign}_0, \ldots, \mathsf{PBS.Sign}_n)$ and $\mathsf{PBS.User} = (\mathsf{PBS.User}_0, \ldots, \mathsf{PBS.User}_m)$ with $m, n \in \mathbb{N}$. The numbers n, m depend on the round of interaction of the protocol. For a generic k-round protocol, the interaction, with b representing a bit, is defined as follows:

$$(\mathsf{msg}_0, \mathsf{st}_0^u) \leftarrow \mathsf{PBS.User}_0(\mathsf{vk}, \varphi, m), (\mathsf{msg}_1, \mathsf{st}_0^s) \leftarrow \mathsf{PBS.Sign}_0(\mathsf{sk}, \varphi, \mathsf{msg}_0),$$
$$(\mathsf{msg}_2, \mathsf{st}_1^u) \leftarrow \mathsf{PBS.User}_1(\mathsf{st}_0^u, \mathsf{msg}_1), (\mathsf{msg}_3, \mathsf{st}_1^s) \leftarrow \mathsf{PBS.Sign}_1(\mathsf{st}_0^s, \mathsf{msg}_2), \ldots,$$
$$(\mathsf{msg}_{2k-1}, b) \leftarrow \mathsf{PBS.Sign}_{k-1}(\mathsf{st}_{k-2}^s, \mathsf{msg}_{2k-2}), \sigma \leftarrow \mathsf{PBS.User}_k(\mathsf{state}_{k-1}^u, \mathsf{msg}_{2k-1})$$

  We write $(0/1, \sigma/\perp) \leftarrow \langle \mathsf{PBS.Sign}(\mathsf{sk}, \varphi), \mathsf{PBS.User}(\mathsf{vk}, \varphi, m)\rangle$ as shorthand for an execution of the above sequence. Following the approach of previous works [HKL19,FPS20,TZ22,FW24,CATZ24], which present blind signature definitions for a fixed round complexity, we define the security properties with a focus on schemes employing 3-round (i.e., 6-message) protocols[9]. Namely, with $\mathsf{PBS.Sign} = (\mathsf{PBS.Sign}_0, \mathsf{PBS.Sign}_1, \mathsf{PBS.Sign}_2)$ and $\mathsf{PBS.User} = (\mathsf{PBS.User}_0, \mathsf{PBS.User}_1, \mathsf{PBS.User}_2, \mathsf{PBS.User}_3)$.
- $\mathsf{PBS.Verify}(\mathsf{vk}, m, \sigma) =: 0/1$ is deterministic and on input a verification key vk, a message m, and a signature σ, outputs 1 if σ is valid on m under vk and 0 otherwise.

A PBS scheme guarantees Correctness, (Strong) One-More Unforgeability and Blindness as defined below hold.

Definition 1 (Correctness). *A predicate blind signature scheme* PBS *for predicate compiler* P *is perfectly correct if for every adversary* A *and* $\lambda \in \mathbb{N}$*:*

$$\Pr\left[\begin{array}{c} m \notin \mathcal{M}_{\mathsf{PBS}} \vee \\ \mathsf{P}(\varphi, m) = 0 \vee \\ (b \wedge b') \end{array} \;\middle|\; \begin{array}{c} \mathsf{par} \leftarrow \mathsf{PBS.Setup}(1^\lambda), (\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{PBS.KeyGen}(\mathsf{par}), \\ (m, \varphi) \leftarrow \mathsf{A}(\mathsf{sk}, \mathsf{vk}), \\ (b, \sigma) \leftarrow \langle \mathsf{PBS.Sign}(\mathsf{sk}, \varphi), \mathsf{PBS.User}(\mathsf{vk}, \varphi, m)\rangle, \\ b' := \mathsf{PBS.Verify}(\mathsf{vk}, m, \sigma) \end{array}\right] = 1$$

Strong One-More Unforgeability (SOMUF). The SOMUF property states that after the completion of ℓ signing sessions with an honest signer, for the malicious user it is hard to compute $\ell + 1$ distinct valid message-signature pairs.

In [FW24] this notion is generalized for PBS schemes. Loosely speaking, the SOMUF property for PBS requires that any valid message-signature pair output by the malicious user after participating in signing sessions with an honest signer, using predicates of its choice, must correspond to a message that satisfies one of the predicates that was actually used during the signing sessions. Specifically, let ℓ be the number of ended sessions, and let φ_j be the predicate used in the j-th session. Suppose the message-signature pairs outputted by the adversary are $(m_i^*, \sigma_i^*)_{i \in [\kappa]}$. The SOMUF property requires the existence of an *injective mapping* $f : [\kappa] \to [\ell]$ so that $\mathsf{P}(\varphi_{f(k)}, m_k^*) = 1$ for all $k \in [\kappa]$.

The one-more unforgeability definition proposed in [FW24] and adopted in this work is strong in two senses. First, it requires that the message-signature pairs outputted by the adversary be distinct (i.e., for all $i_1, i_2 \in [\ell]$, with $i_1 \neq i_2$, it requires that $(m_{i_1}^*, \sigma_{i_1}^*) \neq (m_{i_2}^*, \sigma_{i_2}^*)$). Second, it considers only closed signing sessions (i.e., an open but not finished session never prevents an adversary from forging a signature within that session)[10]. Moreover, the definition in [FW24] considers a predicate valid, and the associated session closed, only if the algorithm $\mathsf{PBS.Sign}_2$ outputs the bit 1. Specifically, a session is "closed", and the predicate is "effectively used in a session" only if $\mathsf{PBS.Sign}_2$ (at the end of the interaction with the malicious user) outputs 1, indicating that the interaction was successful.

[9] Aligning with [FW24], we count a round of communication to be a pair of messages sent between the parties.

[10] In the literature, there exists other works with constructions that are secure, considering only the "non-strong" variant of this definition. For example, the constructions BS_1 and BS_2 from [CATZ24] consider open (but not closed) sessions as "valid" sessions.

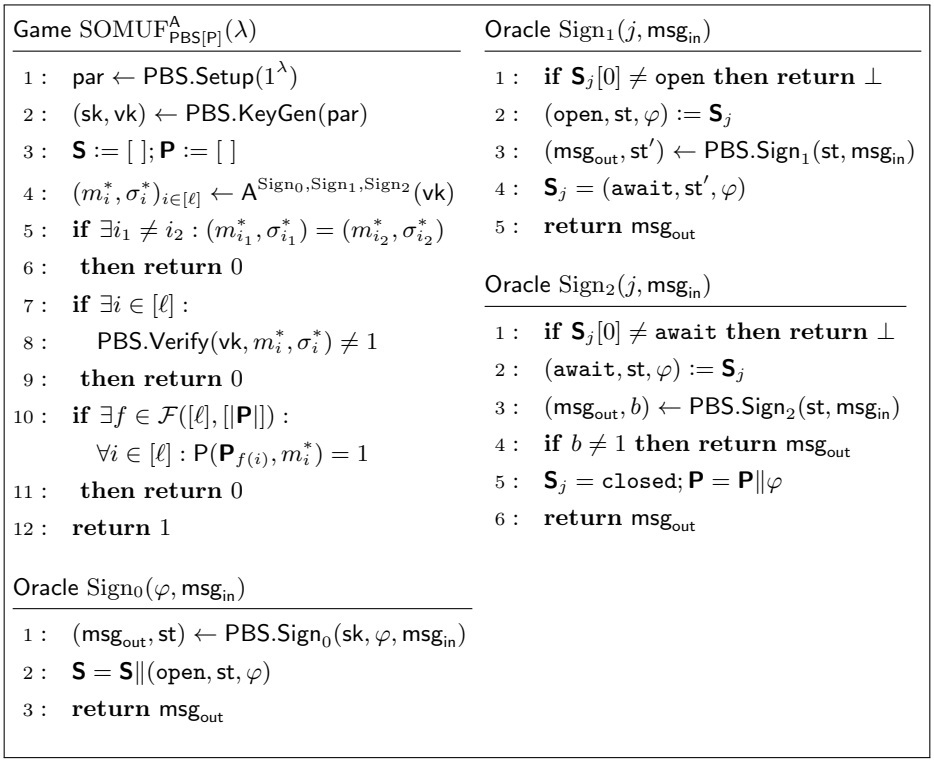


Fig. 1. The SOMUF game for a PBS scheme $\mathsf{PBS[P]}$ with a *2-round or 3-round protocol*. $\mathcal{F}(\mathcal{I}, \mathcal{J})$ denotes the *set* of injective functions from set $\mathcal{I}$ to set $\mathcal{J}$. A "non-strong" version of the SOMUF game is obtained by replacing the first check condition $\exists i_1 \neq i_2 : (m_{i_1}^*, \sigma_{i_1}^*) = (m_{i_2}^*, \sigma_{i_2}^*)$ with $\exists i_1 \neq i_2 : m_{i_1}^* = m_{i_2}^*$.
Note that, in case PBS is 2-round the tuple $(\mathtt{open}, \mathsf{st}, \varphi)$ is replaced with $(\mathtt{await}, \mathsf{st}, \varphi)$.

Definition 2 (Strong One-More Unforgeability). *Consider* SOMUF *described in Fig. 1. A PBS scheme* PBS *for a predicate compiler* P *satisfies strong one-more unforgeability (SOMUF) if for every* PPT *adversary* A *the advantage* $\mathsf{Adv}^{\mathrm{SOMUF}}_{\mathsf{PBS[P]}}(\mathsf{A}, \lambda)$ *is negligible in* λ *where:*

$$\mathsf{Adv}^{\mathrm{SOMUF}}_{\mathsf{PBS[P]}}(\mathsf{A}, \lambda) := \Pr\Big[\mathrm{SOMUF}^{\mathsf{A}}_{\mathsf{PBS[P]}}(\lambda) = 1\Big].$$

Blindness. We recall here the notion of blindness for PBS schemes as in [FW24]. Following their approach, we adopt the same definition of blindness for schemes with parameters, which also covers instantiations without parameters (or with "empty" parameters). Loosely speaking, the blindness property for PBS requires that if a malicious signer interacts with an honest user (or multiple honest users) in n different signing sessions to jointly compute n signatures, then later, when the signer sees one of these n signatures (along with the corresponding message), it cannot identify with more than negligible probability in which session that signature was issued. The definition is given in the *malicious-signer model* [Fis06].

Definition 3 (Blindness). *Consider the game* BLD*, that is, the standard blindness game for blind signature, recalled in Fig. 2. A predicate blind signature scheme* PBS *for a predicate compiler* P *satisfies blindness if for every* PPT *adversary* A *the advantage* $\mathsf{Adv}^{\mathrm{BLD}}_{\mathsf{PBS[P]}}(\mathsf{A}, \lambda)$ *is negligible in* λ *where:*

$$\mathsf{Adv}^{\mathrm{BLD}}_{\mathsf{PBS[P]}}(\mathsf{A}, \lambda) := \Big|\Pr\Big[\mathrm{BLD}^{\mathsf{A},1}_{\mathsf{PBS[P]}}(\lambda) = 1\Big] - \Pr\Big[\mathrm{BLD}^{\mathsf{A},0}_{\mathsf{PBS[P]}}(\lambda) = 1\Big]\Big|$$

When the above distribution ensembles are statistically close, we say that PBS *satisfies* statistical *blindness.*

Note that the blindness notions of [FW24], which follow the classical definitions from previous works, can only ensure the user's privacy if, at the time the user publishes a signature, the signer has blindly signed a sufficiently large number of messages under the same key. In the case of predicate blind signatures, this requires that many different predicates are satisfied by the signed message. Moreover, in the BLD game in Fig. 2, by allowing the adversary A_1 to output distinct predicates φ_0, φ_1, the blindness notion also ensures that the resulting message-signature pair does not reveal any information about the predicate that was used while signing such message.

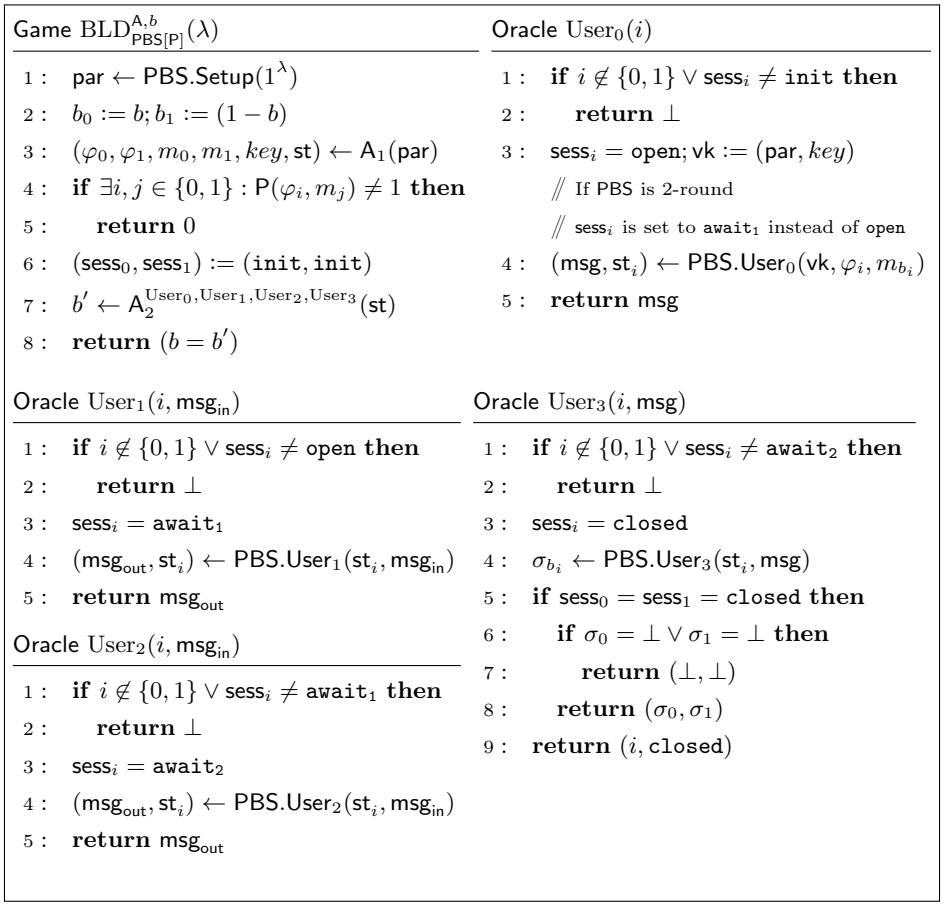


Fig. 2. The BLD game for a predicate blind signature scheme $\mathsf{PBS}[\mathsf{P}]$ with a *2-round or 3-round protocol* with an adversary $\mathsf{A} = (\mathsf{A}_1, \mathsf{A}_2)$, as described in [FW24].

3 (Predicate) Blind Schnorr Signatures

The construction, denoted by PBSch, is described in Fig. 3. Let $\mathcal{M}$ denote the message space of PBSch. Modern Schnorr signatures, as used in practice, are designed to sign arbitrary-length messages, since they rely on cryptographic hash functions with unbounded input length (e.g., SHA-256). In particular, regardless of the message space $\mathcal{M}$ for which one aims to produce blind signatures (e.g., even if restricted to one-bit messages), our blind signature protocol employs the standardized Schnorr scheme, which accepts inputs of arbitrary length. Specifically, within our blind signature protocol, we need to sign messages of the form $(\mathsf{prefix}, \mathsf{suffix})$, where the domain of prefix, denoted by $\mathcal{V}_s$, has super-polynomial cardinality in the security parameter. We denote by $\mathcal{V}_u$ the domain of suffix.

In the security analysis, we explicitly distinguish between different instantiations of the straight-line extractable commitment Cmt used in our construction, depending on the underlying assumption, namely, whether they rely on a CRS or on the NPRO model (see App. A.6). We also carefully decouple the specific hardness assumptions required for each instantiation, and also the corresponding guarantees against different classes of adversaries. Similarly, for ease of exposition, we use a NIWI argument Niwi (see App. A.5).

The design of our construction starts with the protocol of [FW24] presented in Fig. 16, but we deviate in a few crucial steps to obtain improved security and relaxed assumptions. The user sends a commitment C of the message m and the blinding values α, β before receiving the first message $\mathsf{msg}_1 := (R, \nu_s)$ from the signer, where $\nu_s \in \mathcal{V}_s$. Then, the user sends the value $\nu_u \in \mathcal{V}_u$. Finally, after receiving a message from the signer, the user sends a challenge c, a commitment S and a proof. This proof demonstrates either: (a) c was computed from m, α and β and $m \in \mathcal{M}$, or (b) that S is a commitment for two distinct Schnorr signatures $\tilde{\sigma}_0, \tilde{\sigma}_1$ and values $(a, b), (c, d) \in (\mathcal{V}_s \times \mathcal{V}_u) \subseteq \mathcal{M}$, such that $b \neq d$, $c = a$, and $\tilde{\sigma}_0$ is a valid signature for (a, b) while $\tilde{\sigma}_1$ is a valid signature for (c, d). The signer verifies the proof and sends the final message only if the proof is valid. To support predicate blind signatures, the user's proof additionally

asserts that the committed message m satisfies the predicate φ. We define the following parameterized relation $\mathsf{R}_{\mathsf{PBSch}}$:

$$\mathsf{R}_{\mathsf{PBSch}}\left(\begin{array}{l}\mathsf{par}_{\mathsf{R}} := (q, \mathbb{G}, G, \mathsf{H}_q), \mathsf{stm} := (X, X', R, c, C, \varphi, \mathsf{ck}, S),\\ \mathsf{wtn} := (m, \alpha, \beta, \rho, \tilde{\sigma}_0, \tilde{\sigma}_1, \nu_u, \nu'_u, \nu_s, \varrho)\end{array}\right):$$

$$\begin{array}{l}R' := R + \alpha G + \beta X; \mathsf{vk}'_{\mathsf{Sch}} := ((q, \mathbb{G}, G, \mathsf{H}_q), X'); \tilde{m}_0 := (\nu_s, \nu_u); \tilde{m}_1 := (\nu_s, \nu'_u)\\ \textbf{return } (\mathsf{P}(\varphi, m) = 1 \wedge m \in \mathcal{M} \wedge\\ \qquad c = \mathsf{H}_q(R', X, m) + \beta \wedge\ C = \mathsf{Cmt.Com}(\mathsf{ck}, (m, \alpha, \beta); \rho)) \vee\\ \qquad \big(\nu_u \neq \nu'_u \wedge\ S = \mathsf{Cmt.Com}(\mathsf{ck}, (\tilde{\sigma}_0, \tilde{\sigma}_1, \nu_u, \nu'_u, \nu_s); \varrho) \wedge \tilde{m}_0, \tilde{m}_1 \in \mathcal{M} \wedge\\ \qquad \mathsf{Sch.Verify}(\mathsf{vk}'_{\mathsf{Sch}}, \tilde{m}_0, \tilde{\sigma}_0) = 1 \wedge\ \mathsf{Sch.Verify}(\mathsf{vk}'_{\mathsf{Sch}}, \tilde{m}_1, \tilde{\sigma}_1) = 1\big)\end{array}$$

The proof for the relation $\mathsf{R}_{\mathsf{PBSch}}$ is generated using a NIWI argument, according to the scheme Niwi and specifically the algorithm Niwi.Prove, presented in App. A.5. The parameters $\mathsf{par}_{\mathsf{R}}$ used in the first argument of $\mathsf{R}_{\mathsf{PBSch}}$ are the Schnorr signature parameters (Fig. 8). Thus, the Niwi.Rel algorithm simply runs GrGen and obtains $(q, \mathbb{G}, G)$, then runs $\mathsf{HGen}(q)$ and obtains H_q and it outputs the tuple $\mathsf{par}_{\mathsf{R}} := (q, \mathbb{G}, G, \mathsf{H}_q)$. Formalizing the ideas sketched above yields the 3-round PBS scheme PBSch[P, GrGen, HGen, Cmt, Niwi] specified in Fig. 3.

Note that, according to $\mathsf{R}_{\mathsf{PBSch}}$, we require that Cmt can commit tuples of the form (m, α, β). Namely, for all λ, all $\mathsf{par}_{\mathsf{R}} := (q, \mathbb{G}, G, \mathsf{H}_q)$ output by $\mathsf{Niwi.Rel}(1^\lambda)$, all ck output by $\mathsf{Cmt.Setup}(1^\lambda)$, we have $\mathcal{M} \times \mathbb{Z}_q \times \mathbb{Z}_q \subset \mathcal{M}_{\mathsf{Cmt}}$ (where $\mathcal{M}_{\mathsf{Cmt}}$ is the message space of Cmt). Since we use Cmt to commit also to a tuple with two Schnorr signatures, two elements in $\mathcal{V}_u$ and one element in $\mathcal{V}_s$, similarly we assume that $(\mathbb{G} \times \mathbb{Z}_q) \times (\mathbb{G} \times \mathbb{Z}_q) \times \mathcal{V}_u^2 \times \mathcal{V}_s \subset \mathcal{M}_{\mathsf{Cmt}}$.

Correctness. It follows from the one of Niwi and of Sch.

3.1 Security Proofs

Strong One-More Unforgeability. We bound the advantage in breaking the SOMUF (Def. 2) of PBSch by the advantages in breaking the security of the underlying primitives. In Assumption 1, following [FW24], we directly assume SUF-CMA security of the Schnorr signature scheme. We verbatim recall now the motivation behind this choice from [FW24], as it is a critical point in the security analysis. The reason is that all known security proofs of Schnorr signatures are in the RO model [PS96,PS00,FPS20], but the Niwi relation $\mathsf{R}_{\mathsf{PBSch}}$ depends on the used hash function, which would be replaced in the RO model by a random function, for which efficient proofs are not possible. While Assumption 1 may seem unconventional from a theoretical standpoint, we align with [FW24] in arguing that it is practically uncontroversial. Given the widespread use of Schnorr signatures and that it is a *sine qua non* in any application involving Schnorr signatures anyway, we, as in [FW24], assume that the Schnorr signature scheme, instantiated with a standardized cryptographic hash function (e.g., SHA256) as widely available in many real-world applications, is secure.

Theorem 1. *Let* P *be a predicate compiler and* GrGen *and* HGen *be a group and a hash generation algorithm; let* Cmt *be a non-interactive straight-line extractable commitment scheme; let* Sch[GrGen, HGen] *be the Schnorr signature scheme of Fig. 8 instantiated with* GrGen *and* HGen*; and let* Niwi[$\mathsf{R}_{\mathsf{PBSch}}$] *be a NIWI argument system for the relation* $\mathsf{R}_{\mathsf{PBSch}}$*. Then for any* PPT *adversary* A *playing in the game* SOMUF *against the PBS scheme*

PBSch[P, GrGen, HGen, Cmt, Niwi] *defined in Fig. 3 (including the description of* $\mathcal{V}_s$*), successfully completing at most* q *sessions via the oracle* Sign_2*, there exist* PPT *algorithms: (1)* F, C *playing in the game* SUF-CMA *against the unforgeability of* Sch[GrGen, HGen]*, (2)* S *playing in the game* SND *against the soundness of* Niwi[$\mathsf{R}_{\mathsf{PBSch}}$]*, (3)* D *playing in the game* DLOG *against the discrete-logarithm hardness of* GrGen*, (4)* J, K *playing in the game* SLE *against the extractability of* Cmt*, such that for every* $\lambda \in \mathbb{N}$*:*

$$\mathsf{Adv}^{\mathrm{SOMUF}}_{\mathsf{PBS[P]}}(\mathsf{A}, \lambda) \leq \mathsf{Adv}^{\mathrm{SUF\text{-}CMA}}_{\mathsf{Sch[GrGen,HGen]}}(\mathsf{F}, \lambda) + \frac{q^2}{|\mathcal{V}_s|} + \mathsf{Adv}^{\mathrm{SLE}}_{\mathsf{Cmt}}(\mathsf{J}, \lambda) + \mathsf{Adv}^{\mathrm{SND}}_{\mathsf{Niwi}[\mathsf{R}_{\mathsf{PBSch}}]}(\mathsf{S}, \lambda) +$$
$$+ \mathsf{Adv}^{\mathrm{SLE}}_{\mathsf{Cmt}}(\mathsf{K}, \lambda) + q \cdot \exp(1) \cdot \mathsf{Adv}^{\mathrm{DLOG}}_{\mathsf{GrGen}}(\mathsf{D}, \lambda) + \mathsf{Adv}^{\mathrm{SUF\text{-}CMA}}_{\mathsf{Sch[GrGen,HGen]}}(\mathsf{C}, \lambda)$$

The proof of Thm. 1 can be found in App. B. The main idea is to reduce the SOMUF of PBSch to the unforgeability of Schnorr signatures. Given a verification key X, the reduction sets up a second verification key X', the commitment key ck, and answers the adversary's signing queries. When the adversary opens

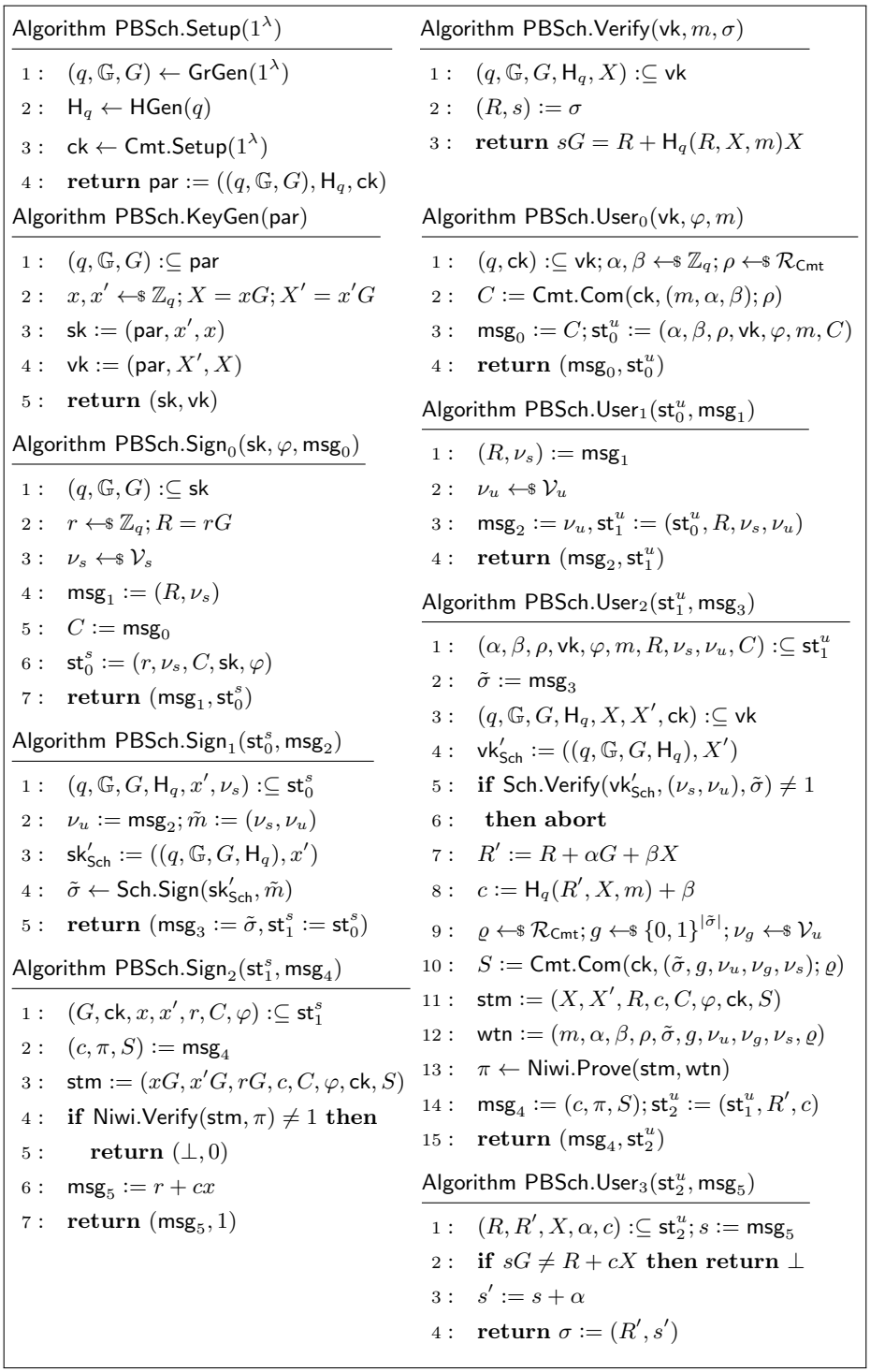


Fig. 3. PBSch[P, GrGen, HGen, Cmt, Niwi], the PBS for Schnorr on a predicate compiler P, a group generation algorithm GrGen, a hash generator HGen, a straight-line extractable commitment Cmt, and a NIWI argument system Niwi for the relation $\mathsf{R}_{\mathsf{PBSch}}$.

a signing session by sending C, the reduction uses the (corresponding) extractor Ext associated with the straight-line extractable commitment Cmt to obtain the committed message composed by m, α, and β. It then queries its own signing oracle for a signature $(\bar{R}, \bar{s})$ on m and sends $R := \bar{R} - \alpha G - \beta X$ to the adversary. Upon receiving c, the proof π attests that it is consistent with m, α, and β, which, by the definition of $\mathsf{R}_{\mathsf{PBSch}}$, implies that $c = \mathsf{H}_q(\bar{R}, X, m) + \beta$. Specifically looking at $\mathsf{R}_{\mathsf{PBSch}}$ we obtain that c is equal to $\mathsf{H}_q(\bar{R}, X, m) + \beta$ because, based on the behavior of the algorithms $\mathsf{PBSch.Sign}_0$ and $\mathsf{PBSch.Sign}_1$, no two signatures are issued under the verification key X' for two messages with the same prefix ν_s but different suffixes (ν_u, ν'_u). Consequently, if an adversary were to obtain such a tuple of message-signature pairs, it would be possible to reduce the attack to the unforgeability of Schnorr signatures under the key X' (i.e., the adversary would have forged one of the two signatures).

Namely, obtaining two signatures on two messages where the suffix of both messages is the same occurs only with the probability of uniformly sampling the same element twice from the set $\mathcal{V}_s$. Since $\mathcal{V}_s$ has super-polynomial cardinality, this probability is negligible. Thus, given that $c = \mathsf{H}_q(\bar{R}, X, m) + \beta$, then by the definition of a Schnorr signature in Fig. 8, let x denote the discrete logarithm of X (i.e., $x := \log_G X$). We have $\bar{s} = \log_G \bar{R} + x\, \mathsf{H}_q(\bar{R}, X, m) = \log_G \bar{R} + x(c - \beta)$. By the definition of PBSch, the adversary expects $s = \log_G R + cx = \log_G \bar{R} - \alpha + x(c - \beta)$, which can be computed as $s = (\bar{s} - \alpha)$.

The proof proceeds via a sequence of hybrid games. In the first hybrid, we extract the messages from the commitment S sent by the adversary, and if it contains two signatures for two messages with the same prefix and different suffixes such that they verify under the verification key X', we terminate the execution. If this is the case, any termination can be used to break the unforgeability of Schnorr with respect to the key X'. In the second hybrid, we also extract the committed values from the commitment C and check whether c is consistent with the plaintext (m, α, β) and whether m satisfies the predicate. If not, we abort the game execution. If this happens, aborts can be used to break the soundness of $\mathsf{Niwi}[\mathsf{R}_{\mathsf{PBSch}}]$. Note that it is possible to break the soundness of $\mathsf{Niwi}[\mathsf{R}_{\mathsf{PBSch}}]$ because the proof π (that breaks the soundness) can only be obtained if no previous termination occurs due to the extraction from the commitment S (according to the first hybrid), that is, the proof has necessarily been computed without having a valid witness for $\mathsf{R}_{\mathsf{PBSch}}$. Then, in the third hybrid, we show that an adversary gains no advantage (in forging a signature) from using the information received from a session that has not been closed (i.e., where the interaction between the adversary and the signer has not successfully concluded), unless it breaks the discrete logarithm assumption. The factor q in the theorem statement comes from a guessing argument following Coron [Cor00], as detailed in App. F according to [FW24]. Finally, we show that for any adversary that can still forge a signature, there is no "explaining" mapping of the adversary's messages to signing sessions, thus the adversary's output must contain a forged Schnorr signature with respect to the key X.

Blindness. The next theorem shows that our protocol PBSch satisfies Def. 3.

Theorem 2. *Let* P *be a predicate compiler,* GrGen *and* HGen *be a group and hash generation algorithm; let* Cmt *be a non-interactive straight-line extractable commitment scheme; and let* $\mathsf{Niwi}[\mathsf{R}_{\mathsf{PBSch}}]$ *be a non-interactive witness indistinguishable argument system for the relation* $\mathsf{R}_{\mathsf{PBSch}}$*. Then for any* PPT *adversary* A *playing in the* BLD *game against the PBS scheme* $\mathsf{PBSch}[\mathsf{P}, \mathsf{GrGen}, \mathsf{HGen}, \mathsf{Cmt}, \mathsf{Niwi}]$ *defined in Fig. 3, there exist* PPT *algorithms: (1)* M_b *and* T_b*, with* $b \in \{0, 1\}$*, playing in the game* HDN *against Hiding of* Cmt*, (2)* W_0 *and* W_1*, playing in the game* WI *against Witness Indistinguishability of* Niwi*, such that for every* $\lambda \in \mathbb{N}$*:*

$$\begin{aligned}\mathsf{Adv}^{\mathrm{BLD}}_{\mathsf{PBSch}}(\mathsf{A}, \lambda) \leq\ & \mathsf{Adv}^{\mathrm{HDN}}_{\mathsf{Cmt}}(\mathsf{M}_0, \lambda) + \mathsf{Adv}^{\mathrm{HDN}}_{\mathsf{Cmt}}(\mathsf{M}_1, \lambda) + \mathsf{Adv}^{\mathrm{WI}}_{\mathsf{Niwi}}(\mathsf{W}_0, \lambda) + \\ & + \mathsf{Adv}^{\mathrm{WI}}_{\mathsf{Niwi}}(\mathsf{W}_1, \lambda) + \mathsf{Adv}^{\mathrm{HDN}}_{\mathsf{Cmt}}(\mathsf{T}_0, \lambda) + \mathsf{Adv}^{\mathrm{HDN}}_{\mathsf{Cmt}}(\mathsf{T}_1, \lambda)\end{aligned}$$

The proof of Thm. 2 can be found in App. C. The proof proceeds via a sequence of hybrid games. In the first hybrid, by rewinding the execution of the adversary, we extract two valid signatures for the verification key X' on two distinct messages in $(\mathcal{V}_s \times \mathcal{V}_u) \subset \mathcal{M}$ that share the same prefix in $\mathcal{V}_s$. In the second hybrid, we replace the message N, which is then committed into S. In the first hybrid, as defined in PBSch in Fig. 3, the message N is a "garbage" message that does not contain two signatures corresponding to two valid messages. In this hybrid, we replace N with a fixed message that is the message extracted by rewinding the execution of the adversary in the first hybrid. Specifically, the message N, now consisting of these two signatures and their corresponding messages, is then used to compute the commitment S for both sessions. By the Hiding of Cmt, we show that this hybrid is indistinguishable from the real game. Then, in the third hybrid, we replace the witness wtn with another valid witness. To give an intuition, such a witness exploits the values in (the fixed) N and satisfies the "second clause

composing the OR" of the relation $\mathsf{R}_{\mathsf{PBSch}}$. By the witness indistinguishability of Niwi, we show that this hybrid is computationally indistinguishable from the second. In the fourth hybrid, we replace the user's commitment C with a commitment to a fixed message. By the Hiding of Cmt, we show that this hybrid is indistinguishable from the second. Finally, using the argument for plain blind Schnorr, we show that this final game is independent of the bit b, which enables us to conclude the proof.

3.2 Instantiation of PBSch

By carefully instantiating the cryptographic components used in our concurrent blind Schnorr signature scheme PBSch, we achieve our main result: a concurrent blind Schnorr signature based only on the security of Schnorr signatures according to Assumption 1, which implies the hardness of the DL problem, while achieving statistical blindness. We formalize this with the following corollary.

Corollary 1. *Let* Cmt *be statistical hiding and* $\mathsf{Niwi}[\mathsf{R}_{\mathsf{PBSch}}]$ *be statistical WI and straight-line extractable, by assuming only the security of Schnorr signatures as stated in Assumption 1, then the protocol* PBSch *realizes a concurrent-secure blind Schnorr signature in the NPRO model with statistical blindness.*

This corollary follows directly from Thms. 1 and 2, together with appropriate instantiations of the cryptographic components used in PBSch. Specifically, we now explain how to instantiate the straight-line extractable commitment scheme to be statistically hiding and the NIWI to be statistically WI and straight-line extractable in the NPRO[11].

We begin with the commitment scheme Cmt. As discussed in Sec. 2 and detailed in App. A, we use Pedersen commitments, which are statistically hiding. To make them straight-line extractable, we also add a straight-line extractable proof of the opening. This proof is constructed starting from the classical (perfect WI) Σ-protocol for proving openings of Pedersen commitments, transformed into a non-interactive version using Fischlin's transform [Fis05]. Further details are given in App. A.

For the second component, we need to instantiate a NIWI that is also statistically WI and straight-line extractable. We describe this construction in two steps: first, we outline how such a construction is possible using Blum's classical Hamiltonicity protocol (though practically inefficient due to expensive NP-reductions), then we explain how the same principles can be adapted to achieve efficient constructions. That is, we start with Blum's classical protocol for Hamiltonicity [Blu87]. For the commitment scheme in the first round, we use the construction from Pass [Pas04], which works as follows: given a message m, it first samples randomness $r \in \{0,1\}^\lambda$, then applies the RO to $m\|r$, using the output as the commitment. The decommitment information is the randomness r.

Two key observations are crucial: (1) Blum's protocol is a Σ-protocol, and (2) the 1-bit challenge version of Blum's protocol is ZK (with constant soundness error). Since ZK implies WI [FS90], it is also WI. WI is preserved under parallel composition [FS90]. Therefore, by executing the 1-bit challenge version of Blum's protocol in parallel, we can reduce the soundness error to negligible while maintaining WI. Thus, Blum's protocol yields a Σ-protocol for Hamiltonicity that is also WI. Given this protocol, we apply the Fiat-Shamir (FS) transform in the NPRO model to Blum's WI Σ-protocol. According to [YZ06, Claim 1], if the original 3-round public-coin HVZK protocol is also WI, the transformed protocol via FS in the NPRO model remains WI. Importantly, the proof of WI does not rely on the random oracle. This establishes that NIWI for NP can indeed be constructed under the only assumption of the existence of a NPRO. We emphasize that this construction is also straight-line extractable according to Def. 13, specifically because the commitments in the first round are straight-line extractable in the NPRO model. This is a well-established result in the literature; see [Pas04, Lemma 7]. Moreover, as shown by Pass in [Pas04, Lemma 9], such commitments are also statistically hiding (based on the output length of the RO). These commitments are efficient to compute in practice, requiring only a single call to the RO for both commitment and decommitment. The main drawback of using Blum's protocol is its practical inefficiency, as proving our relation $\mathsf{R}_{\mathsf{PBSch}}$ requires expensive NP-reductions, but introducing it here can help us to argue in the next paragraph that practically efficient NIWI exists with such characteristics.

That is, a prominent direction for achieving practically usable NIZK arguments is given by MPC-in-the-head approaches [IKOS08], which yield very efficient NIZK [GMO16,**?**,BFH+20,AHIV23]. The security of (some of) these constructions (namely, those listed above) rely only on collision-resistant hash functions (CRHFs). We first note that CRHFs can be constructed similarly to Pedersen commitments,

[11] When using a statistically hiding commitment, standard soundness is insufficient for proving knowledge of the commitment's opening. Therefore, an argument of knowledge is necessary to ensure that the prover actually knows a valid opening.

thus relying on the DL hardness assumption only. For concreteness, we focus on Ligero [AHIV23], and recall the following known observations to show that Ligero can be used for our purposes. In [KLP22, Lemma 7], Kim et al. establish that a (slightly) modified variant of Ligero achieves statistical WI while maintaining the same performance. Based on this result, we can compile this protocol using the Fiat-Shamir transform to obtain an efficient NIWI with statistical WI and performance comparable to Ligero. We are left only with straight-line extractability. Recall that all MPC-in-the-head based constructions require the prover to simulate an MPC among different parties "in his head" and commit to the views of all parties. When such commitments use Pass's commitment scheme leveraging the random oracle, we obtain the necessary extractability property. Note that, while we have highlighted the Ligero-based approach, there exist several other possibilities for obtaining practical NIWI arguments of knowledge in the NPRO model.

A schematic comparison between our PBSch and FWSch [FW24] is presented in Table 1 and a more detailed comparison focusing on efficiency and on the complexity of the relation to be proven compared to the one in [FW24], see App. E.

We conclude by noting that if one wants to avoid assuming the NPRO existence, a construction where there is a CRS composed of only a public key of a PKE scheme can be used. In such a case, we would simply instantiate the extractable commitment with the PKE scheme. However, we have not emphasized such a construction since it does not provide statistical blindness due to the PKE scheme and requires trapdoor permutations.

4 Limits of Game-Based (Predicate) Blind Signatures

Here we identify some limitations in the formalization of (predicate) blind signatures that can turn into security issues. We present a simple counterexample of (predicate) blind signature schemes that satisfies the security definitions from [FW24] (discussed in Sec. 2.1) still maintaining potential vulnerabilities in applications. We first describe a toy application that leverages predicate blind signatures. Next, we show a construction of PBSs that, while secure under the definition of [FW24], trivially compromise the security of the application. Finally, we remark that since PBS are essentially a generalization of blind signatures, this issue can extend to *regular* and *partial* blind signatures as well, specifically in every case where the shared predicate restricts the message space to one that is hard to sample for the signer (i.e., it requires a trapdoor to sample messages from it).

4.1 Hard Predicates

One of the classic motivations for blind signatures originates from Chaum's foundational work [Cha82]. Blind signatures enable electronic cash systems where a bank issues a blind signature as an electronic coin. However, since the bank cannot inscribe the value on the blindly issued coins, it must rely on different public keys for different coin values. This necessitates maintaining and frequently updating a list of public keys, creating significant challenges in different real-world scenarios. Abe and Fujisaki identified this issue in [AF96] and introduced the concept of partially blind signatures to address it. Their approach allowed the signer and user to agree on public information, such as the coin value, as part of the signing process. Fuchsbauer and Wolf later demonstrated that PBS generalizes partially blind signatures [FW24], as reviewed in Sec. 2.1. This generalization leads to PBS extending the capabilities of traditional (partially) blind signatures.

Consider a scenario where a user has a message m that is a signature on a document issued by a bank, confirming that the user owns a certain balance of money. Specifically, m is defined as $m := (\mathsf{pk}, \sigma, m')$, where pk is the public key of the bank, namely the bank that attests to the user's balance, σ is the signature provided by the bank, and m' is the message that declares the ownership of a certain amount of money (e.g., one might think of such a message m' as a bank statement including the current balance, along with personal information such as the user's name and birthdate). The user needs to obtain a signature on the message m from a notary. Note that m' may contain sensitive information (e.g., the user's birthdate) that does not need to be shared with the notary. However, there are other data (e.g., the user's balance in our example) that the notary must be aware of before issuing the signature. Partially blind signatures are seemingly insufficient in the above scenario since part of the message m must remain private while simultaneously providing the notary with guarantees about other parts of m. Instead, PBSs can directly offer a practical solution.

Consider the following predicate compiler P_A. It takes as input the tuple (φ, m), where the predicate is parsed as $(\mathsf{pk}', \mathsf{bal}) := \varphi$, with pk' being the public key of the bank that signed the bank statement for

the user, and $\mathsf{bal} \in \mathbb{N}$ the (required) user's balance. The message m is parsed as $(\mathsf{pk}, \sigma, m') := m$, where pk is the public key of the bank, m' is the message signed by the bank, and σ is a digital signature generated by the signing scheme $\mathsf{Sig} = (\mathsf{Sig.Setup}, \mathsf{Sig.KeyGen}, \mathsf{Sig.Sign}, \mathsf{Sig.Verify})$ (as described in App. A.3). The predicate compiler works as follows:

$$\mathsf{P_A}((\mathsf{pk}', \mathsf{bal}), (\mathsf{pk}, \sigma, m')) = 1 \iff (\mathsf{pk}' = \mathsf{pk} \wedge \mathsf{Sig.Verify}(\mathsf{pk}, m', \sigma) = 1 \wedge \mathsf{bal} :\subseteq {}^{12} m')$$

Using $\mathsf{P_A}$ with a PBS construction clearly realizes the described real-world scenario. The PBS scheme from [FW24], reviewed in Fig. 16, appears well-suited for the above scenario. Specifically, (a) it outputs standard Schnorr signatures, which are widely employed and for which security has been extensively studied, and (b) the definitions of SOMUF and blindness are designed to provide *concurrent security*, allowing the use of such a PBS scheme in complex scenarios where the signer initiates multiple protocol instances with different users concurrently[13].

However, upon closer inspection, the scheme turns out to be vulnerable to a subtle yet significant attack that can be mounted when the predicate is hard, in the sense that for the signer it is hard to find a message such that the predicate is satisfied (as in the real-world scenario presented earlier). It is important to note that this does not affect the correctness of the PBS definition from [FW24] but rather highlights a limitation in its effectiveness, which may be insufficient in certain use cases.

We point out that assuming that the predicate is hard for the signer is not an artificial or merely theoretical assumption. In our real-world example, the notary (the signer) is inherently unable to efficiently find an input that satisfies the predicate. Specifically, the notary cannot sign messages on behalf of the bank, as doing so would directly violate the strong unforgeability of Sig.

Issues on hard predicates. In order to concretely show what can go wrong, due to the limitations of the game-based blindness property, we directly consider the construction proposed in [FW24] of Predicate Blind (Schnorr) Signature (FWSch). Although we have already described the FWSch construction in the Technical Overview, we also formally recall such a construction in Fig. 16. The construction outputs a standard Schnorr signature, according to Fig. 8. As proved in [FW24, Theorem 1 and 2], this construction satisfies the security properties SOMUF and blindness, as presented in Defs. 2 and 3 (under the Assumption 1).

Note that in the given real-world example, the signer (i.e., the notary) initially knows only the predicate (i.e., that it can be satisfied only giving as input tuples of the form $(\mathsf{pk}, \sigma, m')$). However, the notary cannot efficiently find such inputs (i.e., the notary cannot sign on behalf of the bank without violating the unforgeability of Sig).

After successfully interacting with a user, the signer (algorithms $\mathsf{FWSch.Sign}_0, \mathsf{FWSch.Sign}_1$, Fig. 16), gains knowledge of at least one valid message that satisfies the predicate. Thus (in line 6 of the algorithm $\mathsf{FWSch.Sign}_1$ (Fig. 16)) the signer is certain that it has received an accepting proof π from a NIZK argument NArg.

The proof π (despite being zero knowledge) can be used as a witness that a message that satisfies the predicate has been found, meaning that the notary can later reuse it to attest that a user possesses a certain amount of money to others. A crucial observation here, which we will later exploit to address this issue, is that π is generated using a NIZK argument NArg, which inherently lacks deniability[14], ruling out the possibility that the proof was crafted by the signer itself without interacting with a user.

In a nutshell, the signer (i.e., the notary in the example) gains knowledge about the fact that the user knows a valid message satisfying the predicate. Specifically, in the provided example, the notary receives π which is a witness for the user's balance.

Practical relevance of the above issue. Loosely speaking, a malicious signer could exploit this information gained during the interaction by sharing it with third parties or using it for harmful purposes. More in details, looking at the real-world example provided before and the FWSch construction (from Fig. 16) the notary (i.e., the signer of the protocol) can exploit its knowledge of the acceptance of the

[12] We use the notation $\mathsf{bal} :\subseteq m'$ to indicate that the balance bal is recorded in m'. This notation is employed for clarity, since it is irrelevant in our example. However, a more formal approach could be used where a deterministic function checks if the document m' contains the balance bal.

[13] In the real-world scenario, it is likely that the notary interacts with many users simultaneously. Expecting the notary to issue one blind signature at a time is impractical, as this would expose the system to potential DoS attacks.

[14] Deniability refers to the ability to deny having generated a proof, suggesting instead that it could have been simulated by the verifier itself.

proof π to reveal such information to third parties, enabling harmful behaviors (i.e., it can share π to prove that a message from the message space has been sampled). For instance, the notary could leverage this knowledge for financial advantage, such as front-running actions before the (blindly issued) signature is fully exposed (e.g., gaining unfair advantages in financial markets) and damaging the user. Note that, the described scenario clearly suggests a strong analogy to front-running attacks observed in blockchain systems [EMC19]. This indicates that the described vulnerability not only carries theoretical implications but also opens the way for attacks with practical consequences (depending on the protocol in which such blind signature schemes are used).

Fake signers. The Predicate Blind Schnorr Signature FWSch from [FW24], outlined in Fig. 16, requires the use of the secret of the signer only to compute the very last message. Specifically, the secret part of the signing key $x :\subseteq \mathsf{sk}$ is used only in the last round of the protocol. Consider the case of a non-efficiently sampleable message space and a malicious party who does not know the secret key sk but impersonates the signer (in FWSch) from the user's perspective. Such a party can follow the protocol faithfully up to the last round (i.e., executing $\mathsf{FWSch.Sign}_0$) before aborting. The party is not capable of carrying out (only) the final round without knowledge of x. Thus, the party aborts the interaction after receiving the last user's message (before executing the algorithm $\mathsf{FWSch.Sign}_1$). However, this adversary can still collect π (that is in the last user's message of FWSch), as the signer is required to verify π before issuing the message of the last round. Consequently, the malicious party learns the same information as the signer (i.e., a valid message has been sampled from $\mathcal{M}_{\mathsf{PBS}}$). This information is now in the hands of a third party who does not even possess the signature key x.

Severity of issue in the context of PBS. Although we have discussed that such an issue may extend to the case of regular and partial blind signatures, one might rightfully say that this issue is of limited interest, arising only in specific cases where the message space differs from $\{0,1\}^*$, which does not occur frequently. Although this may be true for regular and partial blind signatures, since once the scheme is instantiated, it is defined and runs for a fixed and unchangeable message space, we strongly claim this is *not* the case in the context of PBS [FW24].

Specifically, one can view this primitive as a regular blind signature where the message space is decided dynamically, and thus the scheme can accommodate a potentially different message space in every session. This is because when the parties agree on a shared predicate, they are agreeing on a sub-message space (with respect to the original message space on which the PBS scheme is defined) for which the signature is issued in that session. In other words, the message space is dynamically chosen and depends on the shared predicate, which is a shared input of the parties and thus can change dynamically.

We highlight that this is a particularly relevant issue in the context of PBS because the shared predicate is typically agreed upon by the signer and the user jointly. Thus, a malicious signer can adaptively influence the choice of the shared predicate to define a message space for which sampling is hard for the signer itself, such that any information (even the fact that a sampling has been made) related to the message space gives the signer non-simulatable information.

5 Deniable Blind Signature Schemes

We now present an additional security definition for (predicate) blind signature schemes that captures simulation-based security on the side of the user only. We will consider a definition capturing 3-round protocols only similarly to Sec. 2.1. Deniability requires that the interaction between a malicious signer and a user can be simulated by the malicious signer without ever interacting with the user. This implies that during the interaction, the malicious signer learns no information from the user.

Definition 4 (Deniability). *Consider the games* rDNB *and* sDNB *in Fig. 4. A predicate blind signature scheme* PBS *for a predicate compiler* P *satisfies deniability if there exists an (expected) polynomial time algorithm* $\mathsf{PBS.Sim}$ *such that for any* $\lambda \in \mathbb{N}$*, for any tuple of messages* $\mathsf{msgs} = (m_1, \ldots, m_{\mathsf{poly}(\lambda)})$ *and predicates* $\mathsf{prds} = (\varphi_1, \ldots, \varphi_{\mathsf{poly}(\lambda)})$ *such that* $\mathsf{P}(\varphi_i, m_i) = 1$ *for each* $i \in [\mathsf{poly}(\lambda)]$*, and for every* PPT *adversary* A *we have that for any* PPT *distinguisher* D *the advantage* $\mathsf{Adv}^{\mathrm{rDNB,sDNB}}_{\mathsf{PBS[P]}}(\mathsf{D}, \lambda)$ *is negligible in* λ *where:* $\mathsf{Adv}^{\mathrm{rDNB,sDNB}}_{\mathsf{PBS[P]}}(\mathsf{D}, \lambda) :=$

$$\left|\Pr\left[\mathsf{D}(\mathrm{rDNB}^{\mathsf{A}}_{\mathsf{PBS[P]}}(\lambda, \mathsf{msgs}, \mathsf{prds})) = 1\right] - \Pr\left[\mathsf{D}(\mathrm{sDNB}^{\mathsf{PBS.Sim}}_{\mathsf{PBS[P]}}(\lambda, \mathsf{prds})) = 1\right]\right|$$

A predicate blind signature scheme PBS *satisfies* statistical *deniability if the above holds even for any non-efficient distinguisher.*

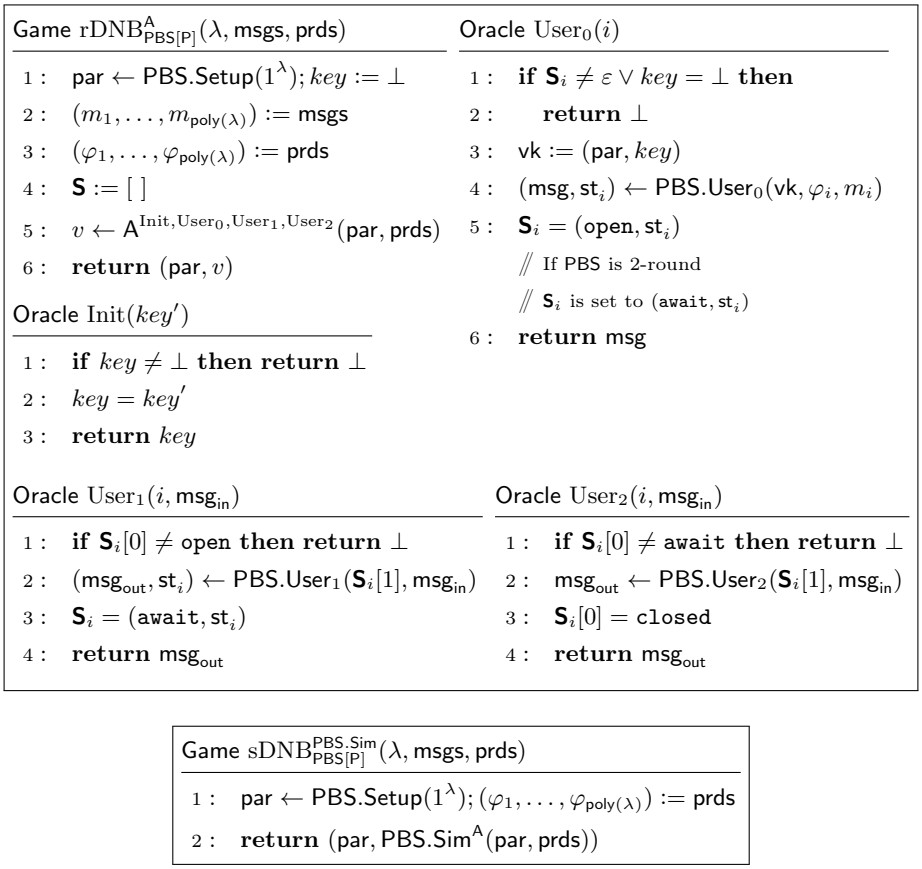


Fig. 4. The rDNB and sDNB games for a predicate blind signature scheme PBS[P] with a *2-round / 3-round protocol* played by the adversary A. Note that PBS.Sim has no access to the tuple of message msgs.

In the rDNB game the adversary A receives the parameters par generated by the PBS.Setup algorithm and the tuple prds. The adversary A can interact with an oracle Init, which, on input key', sets $key = key'$. Notably, the value of key can be set only once per game by A, following the *malicious-signer model* [Fis06]. This verification key (par, key) is used in all the sessions while interacting with oracles User_0, User_1 and User_2 and this is consistent with the definition of the blindness property (Def. 3) for (predicate) blind signature schemes, as adopted in several well-known works from the literature [TZ22,FW24,CATZ24] where, in different sessions, the signer issues blind signatures that are verifiable under one verification key only. In the sDNB game, the simulator PBS.Sim has access to the same information as A in the rDNB game, and only has black-box access to the adversary algorithm A. The outputs of both rDNB and sDNB consist of the parameters par and of the outputs of A or PBS.Sim, respectively. We explicitly include par in the output, following [Pas04, Definition 36], to capture the requirement that the simulator should not choose par (i.e., it cannot be equipped with trapdoor for par or any other additional special power compared to the adversary). Consequently, as in [Pas04], we ensure that the malicious signer does not learn anything beyond the assertion of the statement being proved, including the fact that it does not get a witness proving that a user executed the protocol to get a blind signature.

The above restrictive design in terms of simulator's capabilities addresses issues highlighted in Sec. 4, enforcing deniability. Evidence that a user engaged in a session to get a blind signature could represent a leak of information, as discussed in Sec. 4.1, and our new notion addresses those issues explicitly including par in the output of both rDNB and sDNB, aligning with [Pas04, Definition 36].

Finally, in defining deniability for (predicate) blind signatures, we follow the same spirit as the standard and widely adopted notion of blindness in Def. 3, which is inherently non-adaptive in the sense that the adversary cannot adaptively choose messages based on previously executed sessions. Accordingly, we provide a non-adaptive definition of deniability. This choice is motivated by the same intuition underlying standard blindness definitions. Since the adversary is required to learn nothing from any interaction in a session, there is no meaningful information on which to base adaptive behavior across sessions. Specifically, all sessions must look the same to the adversary, since otherwise she would have an advantage even in the non-adaptive definition of blindness.

While an (arguably contrived) adaptive notion of deniability is in principle feasible, we consider the non-adaptive definition to be adequate and consistent with existing work. Moreover, we believe that the construction presented in this paper already satisfies such an adaptive notion of deniability without requiring any modification.

Relations among properties. We now focus on the Blindness property (Def. 3) and compare it with the Deniability property (Def. 4).

We formalize this distinction with the following theorem, which also formally clarifies the security gaps discussed in Sec. 4.

Theorem 3. *Let* PBS *be a generic (predicate) blind signature scheme where the output message-signature pair has high min-entropy. Then, the notions of* blindness *and* deniability *(as in Defs. 3 and 4) are* incomparable. *Specifically, starting from* PBS*, it is always possible to obtain schemes* PBS_1 *and* PBS_2 *such that: (a)* PBS_1 *satisfy blindness but does not satisfy deniability. (b)* PBS_2 *satisfies deniability but not satisfy blindness.*

We defer the proof to App. G, where we also explicitly (the intuition given in the introduction is, however, sufficient to understand their limitation) discuss why the construction FWSch from [FW24] fails in satisfying the deniability property.

We finally stress that fully-fledged simulation-based blind signatures could not achieve deniability depending on the extra power given to the simulator in the concurrent setting. For instance, a simulator that uses a trapdoor of a programmed CRS or that programs the RO is not executable by the malicious signer in the real world. Moreover, a fully-fledged simulator has the additional (and in some circumstances excessive) requirement of proving also unforgeability through a simulation. In concrete use cases this could represent an excessive overhead damaging the practicality of the construction.

5.1 PBSch satisfies Deniability

The following theorem shows that PBSch satisfies Def. 4.

Theorem 4. *Let* P *be a predicate compiler,* GrGen *and* HGen *be a group and hash generation algorithm; let* Cmt *be a non-interactive straight-line extractable commitment scheme; and let* $\mathsf{Niwi}[\mathsf{R}_{\mathsf{PBSch}}]$ *be a non-interactive witness indistinguishable argument system for the relation* $\mathsf{R}_{\mathsf{PBSch}}$*. Then for any tuple of messages* $\mathsf{msgs} = (m_1, \ldots, m_{\mathsf{poly}(\lambda)})$ *and predicates* $\mathsf{prds} = (\varphi_1, \ldots, \varphi_{\mathsf{poly}(\lambda)})$ *such that* $\mathsf{P}(\varphi_i, m_i) = 1$ *for each* $i \in [\mathsf{poly}(\lambda)]$ *and for any* PPT *adversary* A *playing in* rDNB *game against the PBS scheme* $\mathsf{PBSch}[\mathsf{P}, \mathsf{GrGen}, \mathsf{HGen}, \mathsf{Cmt}, \mathsf{Niwi}]$ *defined in Fig. 3 and an (expected) polynomial time algorithm* $\mathsf{PBSch.Sim}$ *playing in* sDNB *game, there exist* PPT *algorithms: (1)* Q, B *playing in the game* HDN *against Hiding of* Cmt*, (2)* K *playing in the game* WI *against Witness Indistinguishability of* Niwi *such that for every* $\lambda \in \mathbb{N}$*:*

$$\mathsf{Adv}^{\mathrm{rDNB,sDNB}}_{\mathsf{PBSch}[\mathsf{P}]}(\mathsf{D}, \lambda) \leq \mathsf{Adv}^{\mathrm{HDN}}_{\mathsf{Cmt}}(\mathsf{Q}, \lambda) + \mathsf{Adv}^{\mathrm{WI}}_{\mathsf{Niwi}}(\mathsf{K}, \lambda) + \mathsf{Adv}^{\mathrm{HDN}}_{\mathsf{Cmt}}(\mathsf{B}, \lambda)$$

Proof of Thm. 4 can be found in App. D. In the proof, we first show how the simulator $\mathsf{PBSch.Sim}$ works. Loosely speaking, $\mathsf{PBSch.Sim}$ operates by emulating an honest user, with two key exceptions. When the adversary queries the oracle: (a) User_1, the simulator $\mathsf{PBSch.Sim}$ must provide a commitment of the message, but $\mathsf{PBSch.Sim}$ instead commits to a "garbage" value. (b) User_2 (that requires generating the proof π), $\mathsf{PBSch.Sim}$ leverages its black-box access to the adversary A. Specifically, $\mathsf{PBSch.Sim}$ rewinds A to extract a valid witness, to compute the required proof. The core idea underlying $\mathsf{PBSch.Sim}$ rewind strategy is that $\mathsf{PBSch.Sim}$ needs to extract a single tuple from A, in just one session, such that the tuple contains two signatures on messages sharing a common prefix. Once such a tuple is extracted in one session, it suffices for simulating all sessions. $\mathsf{PBSch.Sim}$ runs in expected polynomial time, and both its behavior and analysis closely follow the approach of Canetti et al. [CGGM00], albeit in a simpler setting[15]. The details are delayed to App. D.

The proof proceeds via a sequence of hybrid games that are very similar to the ones in the blindness proof. In the first hybrid, by rewinding the execution of the adversary, we extract two valid signatures for the verification key X' on two distinct messages in $(\mathcal{V}_s \times \mathcal{V}_u) \subset \mathcal{M}$ that share the same prefix in $\mathcal{V}_s$. In the second hybrid, we replace the message N, which is then committed into S. In the first hybrid, as defined in PBSch in Fig. 3, the message N is a "garbage" message that does not contain two signatures

[15] W.r.t. [CGGM00] we deal with concurrent (and not resettable) security, and consider a single verifier rather than polynomially bounded many.

corresponding to two valid messages. In this hybrid, we replace N with a fixed message that is the message extracted by rewinding the execution of the adversary in the first hybrid. Specifically, the message N extracted in one session, consisting of these two signatures and their corresponding messages, is then used in every session to compute the commitment S. By the Hiding of Cmt, we show that this hybrid is indistinguishable from the real game. Then, in the third hybrid, we replace the witness wtn with another valid witness. Such a witness exploits the values in (the fixed for all sessions) N and satisfies the "second clause composing the OR" of the relation $\mathsf{R}_{\mathsf{PBSch}}$. By the witness indistinguishability of Niwi, we show that this hybrid is computationally indistinguishable from the second. In the fourth hybrid, we replace the user's commitment C with a commitment to a fixed message. By the Hiding of Cmt, we show that this hybrid is indistinguishable from the second.

Finally, we prove that the output of the fourth game is statistically indistinguishable from that of the $\mathsf{PBSch.Sim}$ algorithm in the sDNB game, instantiated for the PBS scheme PBSch.

Acknowledgments

We thank the anonymous reviewers of Crypto 2026 for their constructive comments and suggestions.

This work was partially supported by project SERICS (PE00000014) under the MUR National Recovery and Resilience Plan funded by the European Union - NextGenerationEU, and partially supported by the AI-Sec Lab initiative between Sapienza University of Rome and the CYBER 4.0 Competence Center.

All authors are members of the Gruppo Nazionale Calcolo Scientifico-Istituto Nazionale di Alta Matematica (GNCS-INdAM).

References

AF96. Masayuki Abe and Eiichiro Fujisaki. How to date blind signatures. In Kwangjo Kim and Tsutomu Matsumoto, editors, *ASIACRYPT'96*, volume 1163 of *LNCS*, pages 244–251. Springer, Berlin, Heidelberg, November 1996. doi:10.1007/BFb0034851.

AHIV23. Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Ligero: lightweight sublinear arguments without a trusted setup. *DCC*, 91(11):3379–3424, 2023. doi:10.1007/s10623-023-01222-8.

AO00. Masayuki Abe and Tatsuaki Okamoto. Provably secure partially blind signatures. In Mihir Bellare, editor, *CRYPTO 2000*, volume 1880 of *LNCS*, pages 271–286. Springer, Berlin, Heidelberg, August 2000. doi:10.1007/3-540-44598-6_17.

AO09. Masayuki Abe and Miyako Ohkubo. A framework for universally composable non-committing blind signatures. In Mitsuru Matsui, editor, *ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 435–450. Springer, Berlin, Heidelberg, December 2009. doi:10.1007/978-3-642-10366-7_26.

AYY25. Ghous Amjad, Kevin Yeo, and Moti Yung. RSA blind signatures with public metadata. *Proc. Priv. Enhancing Technol.*, 2025(1):37–57, 2025. doi:10.56553/POPETS-2025-0004.

BCC+17. Nir Bitansky, Ran Canetti, Alessandro Chiesa, Shafi Goldwasser, Huijia Lin, Aviad Rubinstein, and Eran Tromer. The hunting of the SNARK. *Journal of Cryptology*, 30(4):989–1066, October 2017. doi:10.1007/s00145-016-9241-9.

BCMS20. Benedikt Bünz, Alessandro Chiesa, Pratyush Mishra, and Nicholas Spooner. Recursive proof composition from accumulation schemes. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part II*, volume 12551 of *LNCS*, pages 1–18. Springer, Cham, November 2020. doi:10.1007/978-3-030-64378-2_1.

BCR+19. Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P. Ward. Aurora: Transparent succinct arguments for R1CS. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part I*, volume 11476 of *LNCS*, pages 103–128. Springer, Cham, May 2019. doi:10.1007/978-3-030-17653-2_4.

BDL+12. Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. High-speed high-security signatures. *Journal of Cryptographic Engineering*, 2(2):77–89, September 2012. doi:10.1007/s13389-012-0027-1.

BFH+20. Rishabh Bhadauria, Zhiyong Fang, Carmit Hazay, Muthuramakrishnan Venkitasubramaniam, Tiancheng Xie, and Yupeng Zhang. Ligero++: A new optimized sublinear IOP. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020*, pages 2025–2038. ACM Press, November 2020. doi:10.1145/3372297.3417893.

BFM88. Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications (extended abstract). In *20th ACM STOC*, pages 103–112. ACM Press, May 1988. doi:10.1145/62212.62222.

BHKR25. Nicholas Brandt, Dennis Hofheinz, Michael Klooß, and Michael Reichle. Tightly-secure blind signatures in pairing-free groups, 2025. In the proceedings of ASIACRYPT 2025. URL: https://eprint.iacr.org/2024/2075.

BJY97. Mihir Bellare, Markus Jakobsson, and Moti Yung. Round-optimal zero-knowledge arguments based on any one-way function. In Walter Fumy, editor, *EUROCRYPT'97*, volume 1233 of *LNCS*, pages 280–305. Springer, Berlin, Heidelberg, May 1997. doi:10.1007/3-540-69053-0_20.

BLL+21. Fabrice Benhamouda, Tancrède Lepoint, Julian Loss, Michele Orrù, and Mariana Raykova. On the (in)security of ROS. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part I*, volume 12696 of *LNCS*, pages 33–53. Springer, Cham, October 2021. doi:10.1007/978-3-030-77870-5_2.

BLS01. Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. In Colin Boyd, editor, *ASIACRYPT 2001*, volume 2248 of *LNCS*, pages 514–532. Springer, Berlin, Heidelberg, December 2001. doi:10.1007/3-540-45682-1_30.

Blu87. Manuel Blum. How to prove a theorem so no one else can claim it. In *Proceedings of the International Congress of Mathematicians*, pages 1444–1451, 1987.

BNPS03. Mihir Bellare, Chanathip Namprempre, David Pointcheval, and Michael Semanko. The one-more-RSA-inversion problems and the security of Chaum's blind signature scheme. *Journal of Cryptology*, 16(3):185–215, June 2003. doi:10.1007/s00145-002-0120-1.

Bol03. Alexandra Boldyreva. Threshold signatures, multisignatures and blind signatures based on the gap-Diffie-Hellman-group signature scheme. In Yvo Desmedt, editor, *PKC 2003*, volume 2567 of *LNCS*, pages 31–46. Springer, Berlin, Heidelberg, January 2003. doi:10.1007/3-540-36288-6_3.

BOV03. Boaz Barak, Shien Jin Ong, and Salil P. Vadhan. Derandomization in cryptography. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 299–315. Springer, Berlin, Heidelberg, August 2003. doi:10.1007/978-3-540-45146-4_18.

BZ23. Paulo L. Barreto and Gustavo H. M. Zanon. Blind signatures from zero-knowledge arguments. Cryptology ePrint Archive, Report 2023/067, 2023. URL: https://eprint.iacr.org/2023/067.

CAHL+22. Rutchathon Chairattana-Apirom, Lucjan Hanzlik, Julian Loss, Anna Lysyanskaya, and Benedikt Wagner. PI-cut-choo and friends: Compact blind signatures via parallel instance cut-and-choose and more. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part III*, volume 13509 of *LNCS*, pages 3–31. Springer, Cham, August 2022. doi:10.1007/978-3-031-15982-4_1.

CATZ24. Rutchathon Chairattana-Apirom, Stefano Tessaro, and Chenzhi Zhu. Pairing-free blind signatures from CDH assumptions. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part I*, volume 14920 of *LNCS*, pages 174–209. Springer, Cham, August 2024. doi:10.1007/978-3-031-68376-3_6.

CDS94. Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In Yvo Desmedt, editor, *CRYPTO'94*, volume 839 of *LNCS*, pages 174–187. Springer, Berlin, Heidelberg, August 1994. doi:10.1007/3-540-48658-5_19.

CGGM00. Ran Canetti, Oded Goldreich, Shafi Goldwasser, and Silvio Micali. Resettable zero-knowledge (extended abstract). In *32nd Annual ACM Symposium on Theory of Computing*, pages 235–244, Portland, OR, USA, May 21–23, 2000. ACM Press. URL: https://eprint.iacr.org/1999/022, doi:10.1145/335305.335334.

Cha82. David Chaum. Blind signatures for untraceable payments. In David Chaum, Ronald L. Rivest, and Alan T. Sherman, editors, *CRYPTO'82*, pages 199–203. Plenum Press, New York, USA, 1982. doi:10.1007/978-1-4757-0602-4_18.

CHYC05. Sherman S. M. Chow, Lucas Chi Kwong Hui, Siu-Ming Yiu, and K. P. Chow. Two improved partially blind signature schemes from bilinear pairings. In Colin Boyd and Juan Manuel González Nieto, editors, *ACISP 05*, volume 3574 of *LNCS*, pages 316–328. Springer, Berlin, Heidelberg, July 2005. doi:10.1007/11506157_27.

CJS14. Ran Canetti, Abhishek Jain, and Alessandra Scafuro. Practical UC security with a global random oracle. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *ACM CCS 2014*, pages 597–608. ACM Press, November 2014. doi:10.1145/2660267.2660374.

CKM+23. Elizabeth C. Crites, Chelsea Komlo, Mary Maller, Stefano Tessaro, and Chenzhi Zhu. Snowblind: A threshold blind signature in pairing-free groups. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part I*, volume 14081 of *LNCS*, pages 710–742. Springer, Cham, August 2023. doi:10.1007/978-3-031-38557-5_23.

CKPR01. Ran Canetti, Joe Kilian, Erez Petrank, and Alon Rosen. Black-box concurrent zero-knowledge requires omega (log n) rounds. In *33rd ACM STOC*, pages 570–579. ACM Press, July 2001. doi:10.1145/380752.380852.

CL24. Yi-Hsiu Chen and Yehuda Lindell. Optimizing and implementing fischlin's transform for UC-secure zero knowledge. *CiC*, 1(2):11, 2024. doi:10.62056/a66chey6b.

Cor00. Jean-Sébastien Coron. On the exact security of full domain hash. In Mihir Bellare, editor, *CRYPTO 2000*, volume 1880 of *LNCS*, pages 229–235. Springer, Berlin, Heidelberg, August 2000. doi:10.1007/3-540-44598-6_14.

COS20. Alessandro Chiesa, Dev Ojha, and Nicholas Spooner. Fractal: Post-quantum and transparent recursive proofs from holography. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part I*, volume 12105 of *LNCS*, pages 769–793. Springer, Cham, May 2020. doi:10.1007/978-3-030-45721-1_27.

CP93. David Chaum and Torben P. Pedersen. Wallet databases with observers. In Ernest F. Brickell, editor, *CRYPTO'92*, volume 740 of *LNCS*, pages 89–105. Springer, Berlin, Heidelberg, August 1993. doi:10.1007/3-540-48071-4_7.

DJW23. Frank Denis, Frederic Jacobs, and Christopher A. Wood. RSA Blind Signatures. RFC 9474, October 2023. doi:10.17487/RFC9474.

DN00. Cynthia Dwork and Moni Naor. Zaps and their applications. In *41st FOCS*, pages 283–293. IEEE Computer Society Press, November 2000. doi:10.1109/SFCS.2000.892117.

EMC19. Shayan Eskandari, Seyedehmahsa Moosavi, and Jeremy Clark. SoK: Transparent dishonesty: Front-running attacks on blockchain. In Andrea Bracciali, Jeremy Clark, Federico Pintore, Peter B. Rønne, and Massimiliano Sala, editors, *FC 2019 Workshops*, volume 11599 of *LNCS*, pages 170–189. Springer, Cham, February 2019. doi:10.1007/978-3-030-43725-1_13.

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. doi:10.1007/11535218_10.

Fis06. Marc Fischlin. Round-optimal composable blind signatures in the common reference string model. In Cynthia Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 60–77. Springer, Berlin, Heidelberg, August 2006. doi:10.1007/11818175_4.

FPS20. Georg Fuchsbauer, Antoine Plouviez, and Yannick Seurin. Blind Schnorr signatures and signed ElGamal encryption in the algebraic group model. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part II*, volume 12106 of *LNCS*, pages 63–95. Springer, Cham, May 2020. doi:10.1007/978-3-030-45724-2_3.

FS90. Uriel Feige and Adi Shamir. Witness indistinguishable and witness hiding protocols. In *22nd ACM STOC*, pages 416–426. ACM Press, May 1990. doi:10.1145/100216.100272.

FW24. Georg Fuchsbauer and Mathias Wolf. Concurrently secure blind Schnorr signatures. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part II*, volume 14652 of *LNCS*, pages 124–160. Springer, Cham, May 2024. doi:10.1007/978-3-031-58723-8_5.

GGS15. Vipul Goyal, Divya Gupta, and Amit Sahai. Concurrent secure computation via non-black box simulation. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 23–42. Springer, Berlin, Heidelberg, August 2015. doi:10.1007/978-3-662-48000-7_2.

GKO+23. Chaya Ganesh, Yashvanth Kondi, Claudio Orlandi, Mahak Pancholi, Akira Takahashi, and Daniel Tschudi. Witness-succinct universally-composable SNARKs. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part II*, volume 14005 of *LNCS*, pages 315–346. Springer, Cham, April 2023. doi:10.1007/978-3-031-30617-4_11.

GKR+21. Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. Poseidon: A new hash function for zero-knowledge proof systems. In Michael Bailey and Rachel Greenstadt, editors, *USENIX Security 2021*, pages 519–535. USENIX Association, August 2021. URL: https://www.usenix.org/conference/usenixsecurity21/presentation/grassi.

GMO16. Irene Giacomelli, Jesper Madsen, and Claudio Orlandi. ZKBoo: Faster zero-knowledge for Boolean circuits. In Thorsten Holz and Stefan Savage, editors, *USENIX Security 2016*, pages 1069–1083. USENIX Association, August 2016. URL: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/giacomelli.

GOS06. Jens Groth, Rafail Ostrovsky, and Amit Sahai. Non-interactive zaps and new techniques for NIZK. In Cynthia Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 97–111. Springer, Berlin, Heidelberg, August 2006. doi:10.1007/11818175_6.

Gro16. Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 305–326. Springer, Berlin, Heidelberg, May 2016. doi:10.1007/978-3-662-49896-5_11.

GWC19. Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. PLONK: Permutations over Lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive, Report 2019/953, 2019. URL: https://eprint.iacr.org/2019/953.

HKL19. Eduard Hauck, Eike Kiltz, and Julian Loss. A modular treatment of blind signatures from identification schemes. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part III*, volume 11478 of *LNCS*, pages 345–375. Springer, Cham, May 2019. doi:10.1007/978-3-030-17659-4_12.

HLW23. Lucjan Hanzlik, Julian Loss, and Benedikt Wagner. Rai-choo! Evolving blind signatures to the next level. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part V*, volume 14008 of *LNCS*, pages 753–783. Springer, Cham, April 2023. doi:10.1007/978-3-031-30589-4_26.

Ide. Iden3. Circom 2: A domain-specific language for defining arithmetic circuits that can be used to generate zero-knowledge proofs. URL: https://docs.circom.io.

IKOS08. Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Cryptography with constant computational overhead. In Richard E. Ladner and Cynthia Dwork, editors, *40th ACM STOC*, pages 433–442. ACM Press, May 2008. doi:10.1145/1374376.1374438.

KLLQ24. Shuichi Katsumata, Yi-Fu Lai, Jason T. LeGrow, and Ling Qin. CSI-otter: isogeny-based (partially) blind signatures from the class group action with a twist. *DCC*, 92(11):3587–3643, 2024. doi:10.1007/s10623-024-01441-7.

KLP22. Allen Kim, Xiao Liang, and Omkant Pandey. A new approach to efficient non-malleable zero-knowledge. In Yevgeniy Dodis and Thomas Shrimpton, editors, *Advances in Cryptology – CRYPTO 2022, Part IV*, volume 13510 of *Lecture Notes in Computer Science*, pages 389–418, Santa Barbara, CA, USA, August 15–18, 2022. Springer, Cham, Switzerland. URL: https://eprint.iacr.org/2022/767, doi:10.1007/978-3-031-15985-5_14.

KNR24. Julia Kastner, Ky Nguyen, and Michael Reichle. Pairing-free blind signatures from standard assumptions in the ROM. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part I*, volume 14920 of *LNCS*, pages 210–245. Springer, Cham, August 2024. doi:10.1007/978-3-031-68376-3_7.

KO21. Stephan Krenn and Michele Orrù. Proposal: Sigma-protocols. In *ZKProof Standards: Workshop 4*, 2021. URL: https://docs.zkproof.org/pages/standards/accepted-workshop4/proposal-sigma.pdf.

KR25. Michael Klooß and Michael Reichle. Blind signatures from proofs of inequality, 2025. In the proceedings of CRYPTO 2025. URL: https://eprint.iacr.org/2024/2076.

KRW24. Michael Klooß, Michael Reichle, and Benedikt Wagner. Practical blind signatures in pairing-free groups. In Kai-Min Chung and Yu Sasaki, editors, *ASIACRYPT 2024, Part I*, volume 15484 of *LNCS*, pages 363–395. Springer, Singapore, December 2024. doi:10.1007/978-981-96-0875-1_12.

Ks22. Yashvanth Kondi and abhi shelat. Improved straight-line extraction in the random oracle model with applications to signature aggregation. In Shweta Agrawal and Dongdai Lin, editors, *ASIACRYPT 2022, Part II*, volume 13792 of *LNCS*, pages 279–309. Springer, Cham, December 2022. doi:10.1007/978-3-031-22966-4_10.

Lin03. Yehuda Lindell. Bounded-concurrent secure two-party computation without setup assumptions. In *35th ACM STOC*, pages 683–692. ACM Press, June 2003. doi:10.1145/780542.780641.

MSM+16. Hiraku Morita, Jacob C. N. Schuldt, Takahiro Matsuda, Goichiro Hanaoka, and Tetsu Iwata. On the security of the Schnorr signature scheme and DSA against related-key attacks. In Soonhak Kwon and Aaram Yun, editors, *ICISC 15*, volume 9558 of *LNCS*, pages 20–35. Springer, Cham, November 2016. doi:10.1007/978-3-319-30840-1_2.

Pas03. Rafael Pass. On deniability in the common reference string and random oracle model. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 316–337. Springer, Berlin, Heidelberg, August 2003. doi:10.1007/978-3-540-45146-4_19.

Pas04. Rafael Pass. *Alternative Variants of Zero-Knowledge Proofs*. Licentiate thesis, KTH Royal Institute of Technology, Stockholm, Sweden, 2004. URL: https://kth.diva-portal.org/smash/record.jsf?pid=diva2:9472.

Ped92. Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 129–140. Springer, Berlin, Heidelberg, August 1992. doi:10.1007/3-540-46766-1_9.

PS96. David Pointcheval and Jacques Stern. Security proofs for signature schemes. In Ueli M. Maurer, editor, *EUROCRYPT'96*, volume 1070 of *LNCS*, pages 387–398. Springer, Berlin, Heidelberg, May 1996. doi:10.1007/3-540-68339-9_33.

PS00. David Pointcheval and Jacques Stern. Security arguments for digital signatures and blind signatures. *Journal of Cryptology*, 13(3):361–396, June 2000. doi:10.1007/s001450010003.

Rep25. Repository, 2025. URL: https://github.com/pierpaolodm/Benchmarking-Blind-Schnorr-Signatures.

Sch01. Claus-Peter Schnorr. Security of blind discrete log signatures against interactive attacks. In Sihan Qing, Tatsuaki Okamoto, and Jianying Zhou, editors, *ICICS 01*, volume 2229 of *LNCS*, pages 1–12. Springer, Berlin, Heidelberg, November 2001. doi:10.1007/3-540-45600-7_1.

Set20. Srinath Setty. Spartan: Efficient and general-purpose zkSNARKs without trusted setup. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part III*, volume 12172 of *LNCS*, pages 704–737. Springer, Cham, August 2020. doi:10.1007/978-3-030-56877-1_25.

TZ22. Stefano Tessaro and Chenzhi Zhu. Short pairing-free blind signatures with exponential security. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part II*, volume 13276 of *LNCS*, pages 782–811. Springer, Cham, May / June 2022. doi:10.1007/978-3-031-07085-3_27.

TZ23. Stefano Tessaro and Chenzhi Zhu. Revisiting BBS signatures. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part V*, volume 14008 of *LNCS*, pages 691–721. Springer, Cham, April 2023. doi:10.1007/978-3-031-30589-4_24.

Wag02. David Wagner. A generalized birthday problem. In Moti Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, pages 288–303. Springer, Berlin, Heidelberg, August 2002. doi:10.1007/3-540-45708-9_19.

Wig19. Avi Wigderson. *Mathematics and Computation: A Theory Revolutionizing Technology and Science*. Princeton University Press, October 2019. doi:10.2307/j.ctvckq7xb.

YZ06. Moti Yung and Yunlei Zhao. Interactive zero-knowledge with restricted random oracles. In Shai Halevi and Tal Rabin, editors, *TCC 2006*, volume 3876 of *LNCS*, pages 21–40. Springer, Berlin, Heidelberg, March 2006. doi:10.1007/11681878_2.

A Additional Preliminaries

We adopt the same (or nearly identical) notation as in [FW24] and report it here for clarity and completeness. We use exp(1) to denote Euler's number. Given a $n \in \mathbb{N}^+$, we let $[n]$ be equal to $\{1, \ldots, n\}$. We let $a := b$ denote the declaration of variable a in the current scope and assigning it the value b. The operator $=$, applied for example in $a = b$, denotes either the overloading of variable a's value with b's value, or, if clear from the context, it denotes the boolean comparison between a and b. The notation $|a|$ denotes the bit-length of a.

An empty list is initialized via $\mathbf{a} := [\,]$. A value x is appended to a list $\mathbf{a}$ via $\mathbf{a} = \mathbf{a} \| x$. The size of $\mathbf{a}$, representing the number of elements in the list, is denoted by $|\mathbf{a}|$. We denote the i-th element of $\mathbf{a}$ by $\mathbf{a}_i$. Given the list $\mathbf{a}$, attempts to access a position $i \notin [|\mathbf{a}|]$ return the empty symbol ε. A tuple of elements is denoted as $x := (a, \ldots, z)$ (where $a, \ldots, z$ are the elements of the tuple) and $x[i]$ denotes the i-th element, which we set to ε if it does not exist.

We denote sets by calligraphic capital letters, e.g., $\mathcal{Q}$, and algorithms by Sans Serif typestyle. The notation $|\mathcal{Q}| \in \mathbb{N}$ represents the cardinality of the set $\mathcal{Q}$. Throughout the paper, we denote the security parameter by $\lambda \in \mathbb{N}$. Given a function $\mathsf{negl} : \mathbb{N} \to \mathbb{R}^+$, we say that $\mathsf{negl}(\lambda)$ is negligible in λ (or shortly negligible) if it vanishes faster than the inverse of any polynomial (i.e., for any constant $c > 0$ for sufficiently large λ it holds that $\mathsf{negl}(\lambda) \leq \lambda^{-c}$). An algorithm is said to run in probabilistic polynomial time (PPT), or to be efficient, if it is randomized, and its number of steps is polynomial in the security parameter λ. For a PPT algorithm A, we write $y \leftarrow \mathsf{A}(x)$ to denote a run of A on input x and output y; if A is randomized, then y is a random variable and $\mathsf{A}(x; \rho)$ denotes a run of A on input x and random coins $\rho \in \{0,1\}^*$, namely $y := \mathsf{A}(x; \rho)$. We assume that uniform sampling from $\mathbb{Z}_n$ is possible for any $n \in \mathbb{N}$. We let $q \leftarrow\!\!\$\ \mathcal{Q}$ denote sampling the variable q uniformly from the set $\mathcal{Q}$.

For a random variable X, we write $\Pr[X = x]$ for the probability that X takes a particular value x. A distribution ensemble $X = \{X(\lambda)\}_{\lambda \in \mathbb{N}}$ is an infinite sequence of random variables indexed by the security parameter $\lambda \in \mathbb{N}$. Two distribution ensembles $X = \{X(\lambda)\}_{\lambda \in \mathbb{N}}$ and $Y = \{Y(\lambda)\}_{\lambda \in \mathbb{N}}$ are said to be *computationally indistinguishable*, denoted by $X \overset{\mathrm{c}}{\approx} Y$, if for every non-uniform PPT algorithm D there exists a negligible function $\mathsf{negl}(\lambda)$ such that: $|\Pr[\mathsf{D}(X(\lambda)) = 1] - \Pr[\mathsf{D}(Y(\lambda)) = 1]| \leq \mathsf{negl}(\lambda)$. When the above holds for all (even unbounded) distinguishers D, we say that X and Y are statistically close.

To enhance the readability of pseudocode, if a value a "implicitly defines" values $(b_1, b_2, \ldots)$ (that is, these can be parsed or anyway trivially obtained from a), we write $(b_1, b_2, \ldots) :\subseteq a$.

We denote by GrGen a PPT algorithm called *group parameter generator* that takes an input 1^λ and outputs $(q, \mathbb{G}, G)$, where $\mathbb{G}$ is a cyclic group of λ-bit prime order q, and G is a generator of $\mathbb{G}$. We implicitly assume that standard group operations in $\mathbb{G}$ can be performed in time polynomial in λ and adopt additive notation. We will often compute over the finite field $\mathbb{Z}_q$ (for a prime q) and do not write modular reduction explicitly when it is clear from the context. Also, we write $x = \log_G X \in \mathbb{Z}_q$ for a group element $X \in \mathbb{G}$ where $X = xG$. Similarly to [FW24] we define a (target-range) *hash function generator* HGen, that is a PPT algorithm that takes as input a number $n \in \mathbb{N}^+$ and returns the description of a function $\mathsf{H}_n : \{0,1\}^* \to \mathbb{Z}_n$. If more values are passed to H_n (e.g., $\mathsf{H}_n(a, \ldots, z)$), they are interpreted as binary strings, concatenated, and then treated as a single value.

A.1 Cryptographic Assumptions

In this paper, we rely on the assumed hardness of the *discrete logarithm* (DL) problem. To capture the DL hardness assumption, for every PPT adversary A, we define the advantage of A playing the game DLOG, described in Fig. 5 as

$$\mathsf{Adv}^{\mathrm{DLOG}}_{\mathsf{GrGen}}(\mathsf{A}, \lambda) := \Pr\left[\mathrm{DLOG}^{\mathsf{A}}_{\mathsf{GrGen}}(\lambda) = 1\right]$$

That is, we assume that $\mathsf{Adv}^{\mathrm{DLOG}}_{\mathsf{GrGen}}(\mathsf{A}, \lambda)$ is negligible in λ.

A.2 Public-Key Encryption Schemes

A public-key encryption (PKE) scheme is a tuple of PPT algorithms $\mathsf{PKE} = (\mathsf{PKE.KeyGen}, \mathsf{PKE.Enc}, \mathsf{PKE.Dec})$, where:

Game $\mathrm{DLOG}^{\mathsf{A}}_{\mathsf{GrGen}}(\lambda)$	
1 :	$(q, \mathbb{G}, G) \leftarrow \mathsf{GrGen}(1^\lambda)$
2 :	$x \leftarrow\!\!\$\ \mathbb{Z}_q; X := xG$
3 :	$y \leftarrow \mathsf{A}(q, \mathbb{G}, G, X)$
4 :	**return** $(x = y)$

Fig. 5. The DLOG game.

- $\mathsf{PKE.KeyGen}(1^\lambda) \to (\mathsf{ek}, \mathsf{dk})$ on input the security parameter, outputs an encryption key ek and a decryption key dk, where ek defines the message space $\mathcal{M}_{\mathsf{PKE}}$, the randomness space $\mathcal{R}_{\mathsf{PKE}}$, and the ciphertext space $\mathcal{C}_{\mathsf{PKE}}$.
- $\mathsf{PKE.Enc}(\mathsf{ek}, m) \to c$ on input an encryption key ek, a message m, where the internal algorithm randomness is sampled uniformly from the set $\mathcal{R}_{\mathsf{PKE}}$, outputs a ciphertext $c \in \mathcal{C}_{\mathsf{PKE}}$ if $m \in \mathcal{M}_{\mathsf{PKE}}$ and $\perp$ otherwise.
- $\mathsf{PKE.Dec}(\mathsf{dk}, c) =: (m/\perp)$ is deterministic and on input a ciphertext $c \in \mathcal{C}_{\mathsf{PKE}}$ and the decryption key dk outputs a message $m \in \mathcal{M}_{\mathsf{PKE}}$ if $c \in \mathcal{C}_{\mathsf{PKE}}$ and $\perp$ otherwise.

and such that Correctness as defined below hold.

Definition 5 (Correctness). *A public-key encryption scheme* PKE *is perfectly correct if for all* $\lambda \in \mathbb{N}$ *and* $m \in \mathcal{M}_{\mathsf{PKE}}$*:*

$$\Pr\left[m = \mathsf{PKE.Dec}(\mathsf{dk}, c) \;\middle|\; (\mathsf{ek}, \mathsf{dk}) \leftarrow \mathsf{PKE.KeyGen}(1^\lambda), c \leftarrow \mathsf{PKE.Enc}(\mathsf{ek}, m)\right] = 1$$

A public-key encryption scheme is said to be CPA-secure if the following definition holds.

Definition 6 (CPA-security). *Consider the game* CPA*, described in Fig. 6. A public-key encryption scheme* PKE *is secure against chosen-plaintext attacks (CPA-secure) if for every* PPT *adversary* A *the advantage* $\mathsf{Adv}^{\mathrm{CPA}}_{\mathsf{PKE}}(\mathsf{A}, \lambda)$ *is negligible in* λ *where:*

$$\mathsf{Adv}^{\mathrm{CPA}}_{\mathsf{PKE}}(\mathsf{A}, \lambda) := \left|\Pr\left[\mathrm{CPA}^{\mathsf{A},0}_{\mathsf{PKE}}(\lambda) = 1\right] - \Pr\left[\mathrm{CPA}^{\mathsf{A},1}_{\mathsf{PKE}}(\lambda) = 1\right]\right|$$

Game $\mathrm{CPA}^{\mathsf{A},b}_{\mathsf{PKE}}(\lambda)$		Oracle $\mathrm{Enc}(m_0, m_1)$	
1 :	$(\mathsf{ek}, \mathsf{dk}) \leftarrow \mathsf{PKE.KeyGen}(1^\lambda)$	1 :	$c \leftarrow \mathsf{PKE.Enc}(\mathsf{ek}, m_b)$
2 :	$b' \leftarrow \mathsf{A}^{\mathrm{Enc}}(\mathsf{ek})$	2 :	**return** c
3 :	**return** $(b = b')$		

Fig. 6. The CPA game.

A.3 Signature Schemes

A signature scheme is a tuple of efficient algorithms $\mathsf{Sig} = (\mathsf{Sig.Setup}, \mathsf{Sig.KeyGen}, \mathsf{Sig.Sign}, \mathsf{Sig.Verify})$ where:

- $\mathsf{Sig.Setup}(1^\lambda) \to \mathsf{sp}$ on input the security parameter, outputs signature parameters sp, which defines the message space $\mathcal{M}_{\mathsf{Sig}}$.
- $\mathsf{Sig.KeyGen}(\mathsf{sp}) \to (\mathsf{sk}, \mathsf{vk})$ on input parameters sp, outputs a signing key sk and a verification key vk.
- $\mathsf{Sig.Sign}(\mathsf{sk}, m) \to \sigma$ on input a signing key sk and a message $m \in \mathcal{M}_{\mathsf{Sig}}$, outputs a signature σ.
- $\mathsf{Sig.Verify}(\mathsf{vk}, m, \sigma) =: 0/1$ is deterministic and on input a verification key vk, a message m and a signature σ, outputs 1 if σ is valid and 0 otherwise.

and such that Correctness and Strong Unforgeability as defined below hold.

Definition 7 (Correctness). *A signature scheme* Sig *is perfectly correct if for all* $\lambda \in \mathbb{N}$ *and* $m \in \mathcal{M}_{\mathsf{Sig}}$*:*

$$\Pr\left[1 = \mathsf{Sig.Verify}(\mathsf{vk}, m, \sigma) \;\middle|\; \begin{array}{c} \mathsf{sp} \leftarrow \mathsf{Sig.Setup}(1^\lambda), \\ (\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{Sig.KeyGen}(\mathsf{sp}), \\ \sigma \leftarrow \mathsf{Sig.Sign}(\mathsf{sk}, m), \end{array}\right] = 1$$

Definition 8 (Strong Unforgeability). *Consider the game* SUF-CMA*, described in Fig. 7. A signature scheme* Sig *satisfies strong existential unforgeability under chosen-message attacks (*SUF-CMA*) if for every* PPT *adversary* A *the advantage* $\mathsf{Adv}^{\text{SUF-CMA}}_{\mathsf{Sig}}(\mathsf{A},\lambda)$ *is negligible in* λ *where:*

$$\mathsf{Adv}^{\text{SUF-CMA}}_{\mathsf{Sig}}(\mathsf{A},\lambda) := \Pr\left[\text{SUF-CMA}^{\mathsf{A}}_{\mathsf{Sig}}(\lambda) = 1\right]$$

Game $\text{SUF-CMA}^{\mathsf{A}}_{\mathsf{Sig}}(\lambda)$	Oracle $\text{Sign}(m)$
1 : $\mathsf{sp} \leftarrow \mathsf{Sig.Setup}(1^\lambda); \mathcal{Q} := \emptyset$	1 : $\sigma \leftarrow \mathsf{Sig.Sign}(\mathsf{sk}, m)$
2 : $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{Sig.KeyGen}(\mathsf{sp})$	2 : $\mathcal{Q} = \mathcal{Q} \cup \{(m, \sigma)\}$
3 : $(m^*, \sigma^*) \leftarrow \mathsf{A}^{\text{Sign}}(\mathsf{vk})$	3 : **return** σ
4 : **return** $((m^*, \sigma^*) \notin \mathcal{Q} \wedge \mathsf{Sig.Verify}(\mathsf{vk}, m^*, \sigma^*) = 1)$	

Fig. 7. The SUF-CMA game.

A.4 Schnorr Signatures

We verbatim recall the Schnorr signature scheme according to [FW24]. The Schnorr signature scheme is

Algorithm $\mathsf{Sch.Setup}(1^\lambda)$	Algorithm $\mathsf{Sch.KeyGen}(\mathsf{sp})$
1 : $(q, \mathbb{G}, G) \leftarrow \mathsf{GrGen}(1^\lambda)$	1 : $(q, \mathbb{G}, G, \mathsf{H}_q) := \mathsf{sp}$
2 : $\mathsf{H}_q \leftarrow \mathsf{HGen}(q)$	2 : $x \leftarrow\!\$\ \mathbb{Z}_q; X := xG$
3 : $\mathsf{sp} := (q, \mathbb{G}, G, \mathsf{H}_q)$	3 : $\mathsf{sk} := (\mathsf{sp}, x); \mathsf{vk} := (\mathsf{sp}, X)$
4 : **return** sp	4 : **return** $(\mathsf{sk}, \mathsf{vk})$
Algorithm $\mathsf{Sch.Sign}(\mathsf{sk}, m)$	**Algorithm** $\mathsf{Sch.Verify}(\mathsf{vk}, m, \sigma)$
1 : $((q, \mathbb{G}, G, \mathsf{H}_q), x) := \mathsf{sk}; r \leftarrow\!\$\ \mathbb{Z}_q$	1 : $((q, \mathbb{G}, G, \mathsf{H}_q), X) := \mathsf{vk}$
2 : $X := xG; R := rG$	2 : $(R, s) := \sigma$
3 : $c := \mathsf{H}_q(R, X, m); s := r + cx$	3 : $c := \mathsf{H}_q(R, X, m)$
4 : **return** $\sigma := (R, s)$	4 : **return** $(sG = R + cX)$

Fig. 8. The Schnorr signature scheme $\mathsf{Sch}[\mathsf{GrGen}, \mathsf{HGen}]$ with key-prefixing based on a group generator GrGen and hash generator HGen.

defined with respect to a *group parameter generator* returning a group of prime order q, and it requires a hash function that maps into $\mathbb{Z}_q$, which can be obtained according to an *hash function generator* with input q.

In Fig. 8 we define Schnorr signatures with "key-prefixing" [BDL+12], which is the variant in use today. Key-prefixing means that the verification key is prepended to the message when signing and verifying (this protects against certain related-key attacks [MSM+16]).

A.5 Non-Interactive Zero-Knowledge

We define a non-interactive zero-knowledge (NIZK) argument/proof system, as defined in [FW24], with respect to a *parameterized relation* $\mathsf{R} : \{0,1\}^* \times \{0,1\}^* \times \{0,1\}^* \to \{0,1\}$ which is a ternary relation that run in polynomial time in the first argument, the parameters, denoted par_R. Given par_R, for a statement stm we call wtn a witness if $\mathsf{R}(\mathsf{par}_\mathsf{R}, \mathsf{stm}, \mathsf{wtn}) = 1$, and define the language $\mathcal{L}_{\mathsf{par}_\mathsf{R}} := \{\mathsf{stm} \mid \exists \mathsf{wtn} : \mathsf{R}(\mathsf{par}_\mathsf{R}, \mathsf{stm}, \mathsf{wtn}) = 1\}$. A NIZK argument system for a relation R is a tuple of efficient PPT algorithms $\mathsf{NArg}[\mathsf{R}] = (\mathsf{NArg.Rel}, \mathsf{NArg.Setup}, \mathsf{NArg.Prove}, \mathsf{NArg.Verify}, \mathsf{NArg.Sim})$ where:

- $\mathsf{NArg.Rel}(1^\lambda) \to \mathsf{par}_\mathsf{R}$ on input the security parameter, returns the relation parameters par_R such that $1^\lambda :\subseteq \mathsf{par}_\mathsf{R}$ and $\mathcal{L}_{\mathsf{par}_\mathsf{R}}$ is an NP-language.
- $\mathsf{NArg.Setup}(\mathsf{par}_\mathsf{R}) \to (\mathsf{crs}, \tau)$ on input relation parameters par_R, returns a common reference string (CRS) crs and a simulation trapdoor τ; the CRS contains the description of par_R, i.e., $\mathsf{par}_\mathsf{R} :\subseteq \mathsf{crs}$.

– $\mathsf{NArg.Prove}(\mathsf{crs}, \mathsf{stm}, \mathsf{wtn}) \to \pi$ on input a CRS crs, a statement stm and a witness wtn, outputs a proof π.
– $\mathsf{NArg.Verify}(\mathsf{crs}, \mathsf{stm}, \pi) =: 0/1$ is deterministic and on input a CRS crs, a statement stm and a proof π, outputs 1 (accept) or 0 (reject).
– $\mathsf{NArg.Sim}(\mathsf{crs}, \tau, \mathsf{stm}) \to \pi$ on input a CRS crs, a simulation trapdoor τ and a statement stm, outputs a proof π.

and such that Correctness, Adaptive Soundness and Zero-Knowledge as defined below hold.

Definition 9 (Correctness). *A NIZK argument system* $\mathsf{NArg}[\mathsf{R}]$ *is perfectly correct if for every tuple* $(\mathsf{par}_\mathsf{R}, \mathsf{stm}, \mathsf{wtn})$*, where* par_R *is output of* $\mathsf{NArg.Rel}$ *with input the security parameter, such that* $\mathsf{R}(\mathsf{par}_\mathsf{R}, \mathsf{stm}, \mathsf{wtn}) = 1$ *and* $\lambda \in \mathbb{N}$*:*

$$\Pr\left[1 = \mathsf{NArg.Verify}(\mathsf{crs}, \mathsf{stm}, \pi) \;\middle|\; \begin{array}{c}(\mathsf{crs}, \tau) \leftarrow \mathsf{NArg.Setup}(\mathsf{par}_\mathsf{R}),\\ \pi \leftarrow \mathsf{NArg.Prove}(\mathsf{crs}, \mathsf{stm}, \mathsf{wtn})\end{array}\right] = 1$$

Definition 10 (Adaptive Soundness). *Consider the game* SND*, described in Fig. 9. A NIZK argument system* $\mathsf{NArg}[\mathsf{R}]$ *is (computationally) adaptively sound if for every* PPT *adversary* A *the advantage* $\mathsf{Adv}^{\mathrm{SND}}_{\mathsf{NArg}[\mathsf{R}]}(\mathsf{A}, \lambda)$ *is negligible in* λ *where:*

$$\mathsf{Adv}^{\mathrm{SND}}_{\mathsf{NArg}[\mathsf{R}]}(\mathsf{A}, \lambda) := \Pr\left[\mathrm{SND}^{\mathsf{A}}_{\mathsf{NArg}[\mathsf{R}]}(\lambda) = 1\right]$$

In case A *is not an efficient algorithm, we refer to such a scheme as a NIZK proof system.*

Game $\mathrm{SND}^{\mathsf{A}}_{\mathsf{NArg}[\mathsf{R}]}(\lambda)$

1 : $\mathsf{par}_\mathsf{R} \leftarrow \mathsf{NArg.Rel}(1^\lambda); (\mathsf{crs}, \tau) \leftarrow \mathsf{NArg.Setup}(\mathsf{par}_\mathsf{R})$
2 : $(\mathsf{stm}, \pi) \leftarrow \mathsf{A}(\mathsf{crs})$
3 : **return** $(1 = \mathsf{NArg.Verify}(\mathsf{crs}, \mathsf{stm}, \pi) \wedge \mathsf{stm} \notin \mathcal{L}_{\mathsf{par}_\mathsf{R}})$

Fig. 9. The SND game, where $\mathcal{L}_{\mathsf{par}_\mathsf{R}} := \{\mathsf{stm} \mid \exists \mathsf{wtn} : \mathsf{R}(\mathsf{par}_\mathsf{R}, \mathsf{stm}, \mathsf{wtn}) = 1\}$.

Definition 11 (Zero Knowledge). *Consider the game* ZK*, described in Fig. 10. A NIZK argument system* $\mathsf{NArg}[\mathsf{R}]$ *is (computationally) zero-knowledge if for every* PPT *adversary* A *the advantage* $\mathsf{Adv}^{\mathrm{ZK}}_{\mathsf{NArg}[\mathsf{R}]}(\mathsf{A}, \lambda)$ *is negligible in* λ *where:*

$$\mathsf{Adv}^{\mathrm{ZK}}_{\mathsf{NArg}[\mathsf{R}]}(\mathsf{A}, \lambda) := \left|\Pr\left[\mathrm{ZK}^{\mathsf{A},0}_{\mathsf{NArg}[\mathsf{R}]}(\lambda) = 1\right] - \Pr\left[\mathrm{ZK}^{\mathsf{A},1}_{\mathsf{NArg}[\mathsf{R}]}(\lambda) = 1\right]\right|$$

Game $\mathrm{ZK}^{\mathsf{A},b}_{\mathsf{NArg}[\mathsf{R}]}(\lambda)$

1 : $\mathsf{par}_\mathsf{R} \leftarrow \mathsf{NArg.Rel}(1^\lambda)$
2 : $(\mathsf{crs}, \tau) \leftarrow \mathsf{NArg.Setup}(\mathsf{par}_\mathsf{R})$
3 : $b' \leftarrow \mathsf{A}^{\mathrm{Prove}}(\mathsf{crs})$
4 : **return** $(b = b')$

Oracle $\mathrm{Prove}(\mathsf{stm}, \mathsf{wtn})$

1 : **if** $\mathsf{R}(\mathsf{par}_\mathsf{R}, \mathsf{stm}, \mathsf{wtn}) \neq 1$ **then**
2 : **return** $\perp$
3 : $\pi_0 \leftarrow \mathsf{NArg.Prove}(\mathsf{crs}, \mathsf{stm}, \mathsf{wtn})$
4 : $\pi_1 \leftarrow \mathsf{NArg.Sim}(\mathsf{crs}, \tau, \mathsf{stm})$
5 : **return** π_b

Fig. 10. The ZK game.

Non-Interactive Witness Indistinguishability. A prerequisite for NIZK systems, as we have presented them, is the CRS. This implies that a NIZK system in the CRS model inherently assumes that the CRS has been generated, according to some distribution, fairly (i.e., by destroying the associated trapdoor). It is possible to overcome such limitations by relaxing the security guarantees of a NIZK system. In particular, instead of requiring the Zero-Knowledge (Def. 11) property, it is possible to rely on the witness indistinguishability (WI) property [FS90]. WI proof/argument systems exist from one-way

functions, and in [FS90], Feige and Shamir demonstrated that they are always secure under concurrent composition (note that this is not always true for ZK proof/argument systems, and it depends on the particular instantiation). The work of Dwork and Naor [DN00] introduced ZAPs, which are two-message public-coin WI proof systems. These proof systems have several advantages: being public-coin, they are publicly verifiable (the validity of the proof can be verified only by looking at the transcript). Furthermore, the first message, which is just a uniformly random string, is inherently reusable for an arbitrary (polynomial) number of proofs on possibly different statements.

Loosely speaking, witness indistinguishability means that an adversary cannot distinguish which one of (at least) two possible witnesses, wtn_1 or wtn_2, was used in generating the proof[16]. Unlike NIZK systems for which we know the impossibility in the plain model [BFM88], and therefore require the CRS model, non-interactive witness indistinguishability (NIWI) argument/proof systems are possible in the plain model. The first NIWI construction in the plain model was proposed by Barak et al. [BOV03]. Later, Groth et al. [GOS06] propose more efficient NIWI construction in the plain model for concrete language.

A NIWI argument system for a relation R is a tuple of PPT algorithms $\mathsf{Niwi}[\mathsf{R}] = (\mathsf{Niwi.Rel}, \mathsf{Niwi.Prove}, \mathsf{Niwi.Verify})$, where, looking at $\mathsf{NArg}[\mathsf{R}]$ defined in App. A.5, the algorithm Niwi.Rel works exactly as the algorithm NArg.Rel, and the same applies to Niwi.Prove and Niwi.Verify, apart from the fact that these algorithms do not take the CRS as input. Since no CRS is provided as input, we do not explicitly indicate that both Niwi.Prove and Niwi.Verify take par_R as input. However, we implicitly assume that these algorithms always run with par_R, which is output by Niwi.Rel. A NIWI argument system satisfies correctness according to Def. 9, with the only difference being that the probability is not conditioned on the setup algorithm. This is because $\mathsf{Niwi}[\mathsf{R}]$ does not include such an algorithm, and the proving and verifying algorithms are now Niwi.Prove and Niwi.Verify, which do not require the CRS as input. A NIWI argument system also satisfies soundness according to Def. 10, with the only difference being that in the soundness game SND (Fig. 9), there is no setup algorithm. Thus, the adversary does not receive the CRS as input but only par_R. Additionally, the verification algorithm is replaced by Niwi.Verify, in line with the modifications discussed for correctness.

Moreover, a NIWI argument system $\mathsf{Niwi}[\mathsf{R}]$ satisfies the witness indistinguishability property, which we formalize in the following definition.

Definition 12 (Witness Indistinguishability). *Consider the game* WI*, described in Fig. 11. A NIWI argument system* $\mathsf{Niwi}[\mathsf{R}]$ *satisfies witness indistinguishability if for every* PPT *adversary* A *the advantage* $\mathsf{Adv}^{\mathrm{WI}}_{\mathsf{Niwi}[\mathsf{R}]}(\mathsf{A}, \lambda)$ *is negligible in* λ *where:*

$$\mathsf{Adv}^{\mathrm{WI}}_{\mathsf{Niwi}[\mathsf{R}]}(\mathsf{A}, \lambda) := \left| \Pr\left[\mathrm{WI}^{\mathsf{A},0}_{\mathsf{Niwi}[\mathsf{R}]}(\lambda) = 1\right] - \Pr\left[\mathrm{WI}^{\mathsf{A},1}_{\mathsf{Niwi}[\mathsf{R}]}(\lambda) = 1\right] \right|$$

In case the above distribution ensambles are statistically close, we say that $\mathsf{Niwi}[\mathsf{R}]$ *satisfies* statistical *witness indistinguishability.*

Game $\mathrm{WI}^{\mathsf{A},b}_{\mathsf{Niwi}[\mathsf{R}]}(\lambda)$	Oracle $\mathrm{Prove}(\mathsf{stm}, \mathsf{wtn}_0, \mathsf{wtn}_1)$
1 : $\mathsf{par}_\mathsf{R} \leftarrow \mathsf{Niwi.Rel}(1^\lambda)$	1 : **if** $\exists \tilde{b} \in \{0,1\} : \mathsf{R}(\mathsf{par}_\mathsf{R}, \mathsf{stm}, \mathsf{wtn}_{\tilde{b}}) \neq 1$ **then**
2 : $b' \leftarrow \mathsf{A}^{\mathrm{Prove}}(\mathsf{par}_\mathsf{R})$	2 : **return** $\perp$
3 : **return** $(b = b')$	3 : $\pi \leftarrow \mathsf{Niwi.Prove}(\mathsf{stm}, \mathsf{wtn}_b)$
	4 : **return** π

Fig. 11. The WI game.

NIWI straight-line extractable. We present a definition of a NIWI that is also an argument of knowledge. Loosely speaking, this means that for any adversary outputting an accepting proof, there exists an extractor machine that can extract the witness with overwhelming probability. We focus on extractors that obtain the witness straight-line (i.e., without rewinding the adversary) in the NPRO model (i.e., the extractor is given access to all queries and answers made to the RO before its execution). The following definition follows the one in [Pas04, Definition 40].

[16] Clearly, such a definition makes sense only when there exists more than one witness for each statement; otherwise, the property is trivially satisfied.

Definition 13 (Straight-Line Extractability). *Consider the game* AOK*, described in Fig. 12, where Γ in the* AOK *game is the list of all the RO queries and answers performed by* A*. A NIWI argument system* $\mathsf{Niwi}[\mathsf{R}] = (\mathsf{Niwi.Prove}, \mathsf{Niwi.Verify})$ *satisfies the straight-line extractability property if a) there exists an extractor machine* Ext *and b) for every* PPT *adversary* A *the advantage* $\mathsf{Adv}^{\mathrm{AOK}}_{\mathsf{Niwi}[\mathsf{R}]}(\mathsf{A}, \lambda)$ *is negligible in* λ *where:* $\mathsf{Adv}^{\mathrm{AOK}}_{\mathsf{Niwi}[\mathsf{R}]}(\mathsf{A}, \lambda) := \Pr\left[\mathrm{AOK}^{\mathsf{A}}_{\mathsf{Niwi}[\mathsf{R}]}(\lambda) = 1\right]$.

Game $\mathrm{AOK}^{\mathsf{A}}_{\mathsf{Niwi}[\mathsf{R}]}(\lambda)$	
1 :	$(\mathsf{stm}, \pi) \leftarrow \mathsf{A}(1^\lambda); \mathsf{wtn} \leftarrow \mathsf{Ext}(\mathsf{stm}, \pi, \Gamma)$
2 :	**return** $(1 = \mathsf{Niwi.Verify}(\mathsf{stm}, \pi) \wedge 1 \neq \mathsf{R}(\mathsf{stm}, \mathsf{wtn})))$

Fig. 12. The AOK game.

A.6 Commitment Schemes

A commitment scheme is a tuple of efficient algorithms $\mathsf{Cmt} = (\mathsf{Cmt.Setup}, \mathsf{Cmt.Com}, \mathsf{Cmt.Dec})$[17], where:

- $\mathsf{Cmt.Setup}(1^\lambda) \to \mathsf{ck}$ on input the security parameter, outputs a commitment key ck, which defines the message space $\mathcal{M}_{\mathsf{Cmt}}$, the randomness space $\mathcal{R}_{\mathsf{Cmt}}$, and the commitment space $\mathcal{C}_{\mathsf{Cmt}}$.
- $\mathsf{Cmt.Com}(\mathsf{ck}, m) \to (c, d)$ on input the commitment key ck and a message m, where the internal algorithm randomness is sampled uniformly from the set $\mathcal{R}_{\mathsf{Cmt}}$, outputs a commitment $c \in \mathcal{C}_{\mathsf{Cmt}}$ and a decommitment information $d \in \mathcal{D}_{\mathsf{Cmt}}$ if $m \in \mathcal{M}_{\mathsf{Cmt}}$ and $\perp$ otherwise.
- $\mathsf{Cmt.Dec}(\mathsf{ck}, c, m, d) =: 0/1$ is deterministic and on input the commitment key ck, a commitment $c \in \mathcal{C}_{\mathsf{Cmt}}$, a message $m \in \mathcal{M}_{\mathsf{Cmt}}$ and a decommitment information $d \in \mathcal{D}_{\mathsf{Cmt}}$, outputs 1 if c is a valid commitment for m, according to d and 0 otherwise.

and such that Correctness, Binding and Hiding as defined below hold.

Definition 14 (Correctness). *A commitment scheme* Cmt *is perfectly correct if for all* $\lambda \in \mathbb{N}$ *and* $m \in \mathcal{M}_{\mathsf{Cmt}}$*:*

$$\Pr\left[1 = \mathsf{Cmt.Dec}(\mathsf{ck}, c, m, d) \;\middle|\; \begin{array}{c} \mathsf{Cmt.Setup}(1^\lambda) \to \mathsf{ck} \\ \mathsf{Cmt.Com}(\mathsf{ck}, m) \to (c, d) \end{array}\right] = 1$$

Definition 15 (Binding). *Consider the game* BND*, described in Fig. 13. A commitment scheme* Cmt *is binding if for every* PPT *adversary* A *the advantage* $\mathsf{Adv}^{\mathrm{BND}}_{\mathsf{Cmt}}(\mathsf{A}, \lambda)$ *is negligible in* λ *where:*

$$\mathsf{Adv}^{\mathrm{BND}}_{\mathsf{Cmt}}(\mathsf{A}, \lambda) := \Pr\left[\mathrm{BND}^{\mathsf{A}}_{\mathsf{Cmt}}(\lambda) = 1\right]$$

Game $\mathrm{BND}^{\mathsf{A}}_{\mathsf{Cmt}}(\lambda)$	
1 :	$\mathsf{ck} \leftarrow \mathsf{Cmt.Setup}(1^\lambda)$
2 :	$(c, m_0, d_0, m_1, d_1) \leftarrow \mathsf{A}(\mathsf{ck})$
3 :	**return** $(m_0 \neq m_1 \wedge \mathsf{Cmt.Dec}(\mathsf{ck}, c, m_0, d_0) = 1 \wedge \mathsf{Cmt.Dec}(\mathsf{ck}, c, m_1, d_1) = 1)$

Fig. 13. The BND game.

Definition 16 (Hiding). *Consider the game* HDN*, described in Fig. 14. A commitment scheme* Cmt *is hiding if for every* PPT *adversary* A *the advantage* $\mathsf{Adv}^{\mathrm{HDN}}_{\mathsf{Cmt}}(\mathsf{A}, \lambda)$ *is negligible in* λ *where:*

$$\mathsf{Adv}^{\mathrm{HDN}}_{\mathsf{Cmt}}(\mathsf{A}, \lambda) := \left|\Pr\left[\mathrm{HDN}^{\mathsf{A},0}_{\mathsf{Cmt}}(\lambda) = 1\right] - \Pr\left[\mathrm{HDN}^{\mathsf{A},1}_{\mathsf{Cmt}}(\lambda) = 1\right]\right|$$

In case the above distribution ensambles are statistically close, we say that Cmt *satisfies* statistical *hiding.*

[17] In this work, we focus only on non-interactive commitment schemes.

Game $\mathrm{HDN}^{\mathsf{A},b}_{\mathsf{Cmt}}(\lambda)$	Oracle $\mathrm{Com}(m_0, m_1)$
1 : $\mathsf{ck} \leftarrow \mathsf{Cmt.Setup}(1^\lambda)$	1 : **if** $\exists \tilde{b} \in \{0,1\} : m_{\tilde{b}} \notin \mathcal{M}_{\mathsf{Cmt}}$ **then**
2 : $b' \leftarrow \mathsf{A}^{\mathrm{Com}}(\mathsf{ck})$	2 : **return** $\perp$
3 : **return** $(b = b')$	3 : $(c, d) \leftarrow \mathsf{Cmt.Com}(\mathsf{ck}, m_b)$
	4 : **return** c

Fig. 14. The HDN game.

Extractable commitment schemes. In the literature, commitment schemes are often extended with an additional guarantee, obtaining what are known as *extractable commitment schemes*. Informally, such schemes are standard commitment schemes (as defined above) with the added property that there exists an (efficient) extractor Ext which, given black-box access to any efficient adversarial sender A that successfully produces a commitment, can extract the unique message that can be successfully decommitted to.

Since we focus exclusively on non-interactive commitment schemes, the extractor must be given additional capabilities (as for zero-knowledge simulators). Note that, without such capabilities, extraction would also be feasible for (malicious) receivers, violating the hiding property. That is, we are interested only in *straight-line extractable* schemes, where extraction proceeds without rewinding, in the non-programmable random oracle (NPRO) model.

Definition 17 ([Pas03] Non-Interactive Straight-Line Extractable Commitment Scheme). *Consider the game* SLE*, described in Fig. 15. A non-interactive commitment scheme* Cmt *is straight-line extractable in the NPRO model if a) there exists a* PPT *extractor* Ext *that can observe queries and answers made to the random oracle, and b) for every* PPT *adversary* A*, the advantage* $\mathsf{Adv}^{\mathrm{SLE}}_{\mathsf{Cmt}}(\mathsf{A}, \lambda)$ *is negligible in* λ *where:*

$$\mathsf{Adv}^{\mathrm{SLE}}_{\mathsf{Cmt}}(\mathsf{A}, \lambda) := \Pr\left[\mathrm{SLE}^{\mathsf{A}}_{\mathsf{Cmt}}(\lambda) = 1\right]$$

Game $\mathrm{SLE}^{\mathsf{A}}_{\mathsf{Cmt}}(\lambda)$
1 : $\mathsf{ck} \leftarrow \mathsf{Cmt.Setup}(1^\lambda)$
2 : $(c, m, d) \leftarrow \mathsf{A}(\mathsf{ck}); m' \leftarrow \mathsf{Ext}(\mathsf{ck}, c)$
3 : **return** $\left(m \neq m' \wedge \mathsf{Cmt.Dec}(\mathsf{ck}, c, m, d) = 1\right)$

Fig. 15. The BND game.

Instantiation of straight-line extractable commitments in the NPRO model. In the NPRO model, simple constructions such as committing to a message m by giving the RO the input (m, r), with $r \leftarrow\!\!\$\ \{0,1\}^*$, and using the output as the commitment, allow for straight-line extraction [Pas03], but do not support proving statements about the committed message, since the RO lacks a meaningful circuit representation. While one could assume a concrete instantiation of the RO admits such a representation [BCMS20,COS20], we avoid this heuristic for the commitment.

Instead, following [GKO+23,KRW24], we adopt so-called *provable* commitments, where the commitment consists of a pair $(c', \pi_{c'})$, in this way c' is a standard-model commitment (with a circuit representation), and $\pi_{c'}$ is a straight-line extractable WI proof of knowledge of the opening of c'.

In this work, we instantiate such straight-line extractable commitment by using a Pedersen commitment for computing c', and constructing the proof $\pi_{c'}$ via the transformations in [Pas04,Fis05] applied to the Σ-protocol for proving knowledge of the opening of a Pedersen commitment. Specifically, the commitment setup first runs the GrGen algorithm and obtains a group $\mathbb{G}$ of order q with generator G, and derives an independent generator H (e.g., by hashing public input into the group via the random oracle). The commitment key is $(q, \mathbb{G}, G, H)$, and assuming the hardness of DL problem in $\mathbb{G}$ we also have that it is infeasible for an efficient adversary to compute $\log_G H$. To commit to $m \in \mathbb{Z}_q$, the algorithm samples $r \leftarrow\!\!\$\ \mathbb{Z}_q$ and sets $c' = mG + rH$; the decommitment information d is r. The full commitment is $c = (c', \pi_{c'})$, where $\pi_{c'}$ proves knowledge of (m, r) such that correctly decommits to c'; we discuss the construction of $\pi_{c'}$ in more detail below. A commitment is considered well-formed if c' is well-formed with respect to the commitment key and $\pi_{c'}$ is accepting. The decommitment algorithm, on input (m, d),

checks whether c' is equal to $mG + dH$, if so, it outputs 1, otherwise 0. We now focus on the instantiation of the proof $\pi_{c'}$. Starting from the Σ-protocol for proving knowledge of the opening of a Pedersen commitment[18], our goal is to obtain a non-interactive, straight-line extractable WI proof of knowledge in the NPRO model.

This transformation is well-established in the literature. We would ideally adopt the transformation of [Fis05] or its randomized variant [Ks22], which also offers practical performance [CL24]. However, these approaches are proven secure (i.e., satisfy ZK) in the *programmable* RO model and do not guarantee prior WI in the NPRO model.

Still, in our setting, we can exploit specific properties of the underlying Σ-protocol. First, the Σ-protocol for the opening of a Pedersen commitment is *perfectly* SHVZK. Then, given the result of [CDS94], we recall that every Σ-protocol that is perfectly SHVZK is also *perfectly* WI. Since the transformations in [Fis05,Ks22] preserve the transcript structure of the underlying Σ-protocol (i.e., the proof consists of all valid transcripts), it follows that the resulting non-interactive proof also inherits perfect WI. Therefore, although WI in the NPRO model is not guaranteed in the general case, for the specific case of the Pedersen commitment, this property holds tightly, and we can safely instantiate $\pi_{c'}$ using the transformation of [Fis05,Ks22].

Note that this instantiation achieves statistical hiding and computational binding assuming the hardness of the DL problem, and its security is established according to the proofs in [GKO+23,Pas04], where the main idea of the proofs follows the approach we discussed above.

A.7 The Concurrent Blind Schnorr Construction from [FW24]

In Fig. 16, we recall the formal description of the concurrent blind Schnorr signature construction from [FW24], which we denote by FWSch.

B Proof of Theorem 1

We give a formal proof that our predicate blind signature scheme PBSch from Fig. 3 satisfies strong one-more unforgeability according to Def. 2 by providing reductions to the security properties of its building blocks. We proceed by a sequence of games specified in Fig. 17. In this section, we employ the keyword "**halt the run with**" inside an oracle to explicitly denote the termination of the game and its output, rather than the return value of the oracle.

For clarity, we begin by outlining the structure of the proof. To simplify the exposition, we first present the proof of the theorem in the case where the commitment Cmt is instantiated directly via a public-key encryption scheme PKE, whose setup safely generates a key pair and whose extraction is performed by decryption. This case is particularly convenient because (a) it is closer to [FW24], thus (b) it lets us focus on the computation of the NIWI argument, which is the core of our construction. We then argue that replacing PKE with a straight-line extractable commitment (e.g., the NPRO-based instantiation discussed above) makes no difference: the reduction remains the same.

Game H_0**.** This is game SOMUF from Fig. 1 with PBS instantiated by PBSch from Fig. 3. The generic $\mathsf{PBS.Setup}$ hence is replaced by the setup from Fig. 3. In Sign_0, the call $\mathsf{PBS.Sign}_0$ is instantiated by sampling $r \leftarrow\!\$\, \mathbb{Z}_q$ and $\nu_s \leftarrow\!\$\, \mathcal{V}_s$; and returning $R = rG$ and ν_s. In Sign_1, the call $\mathsf{PBS.Sign}_1$ is instantiated by signing the message $\tilde{m} := (\nu_s, \nu_u) \in \mathcal{M}$ using $\mathsf{Sch.Sign}$ from Fig. 8; and returning such signature $\tilde{\sigma}$. In Sign_2, the call $\mathsf{PBS.Sign}_2$ is instantiated by the verification of the NIWI argument, and it returns 0 if the verification fails or $(r + cx)$.

Game H_1**.** In the game H_1 we modify Sign_1, so that on each call with input (j, ν_u) the output of the algorithm $\mathsf{Sch.Sign}$, that is $\tilde{\sigma}$, is added to the set $\mathcal{U}$ along with the message $\tilde{m} := (\nu_s, \nu_u)$ signed by such algorithm (i.e., $(\tilde{\sigma}, \tilde{m} = (\nu_s, \nu_u))$ is added to $\mathcal{U}$). Moreover, in each Sign_2 call on input $(j, (c, \pi, S))$, we decrypt S to obtain the values $(\tilde{\sigma}_0, \tilde{\sigma}_1, \nu_1, \nu_1', \nu_0)$, then we check if $\tilde{\sigma}_0$ is a valid signature for the message $\tilde{m}_0 := (\nu_0, \nu_1)$ under the verification key $(q, \mathbb{G}, G, \mathsf{H}_q, X')$ and if $\tilde{\sigma}_1$ is a valid signature for the message $\tilde{m}_1 := (\nu_0, \nu_1')$ under same the verification key, if so we stop the game and return 0.

Note that, because the PKE we use is perfectly correct, we may perform decryption inside the game without introducing an additional reduction or an explicit hybrid: when Cmt is instantiated by this PKE

[18] A Σ-protocol is a 3-round public-coin protocol satisfying special honest-verifier zero-knowledge (SHVZK) and 2-special soundness. For a formal definition, see [KO21] and for the case of the Pedersen commitment, see [KO21, Example 4] and [Ped92].

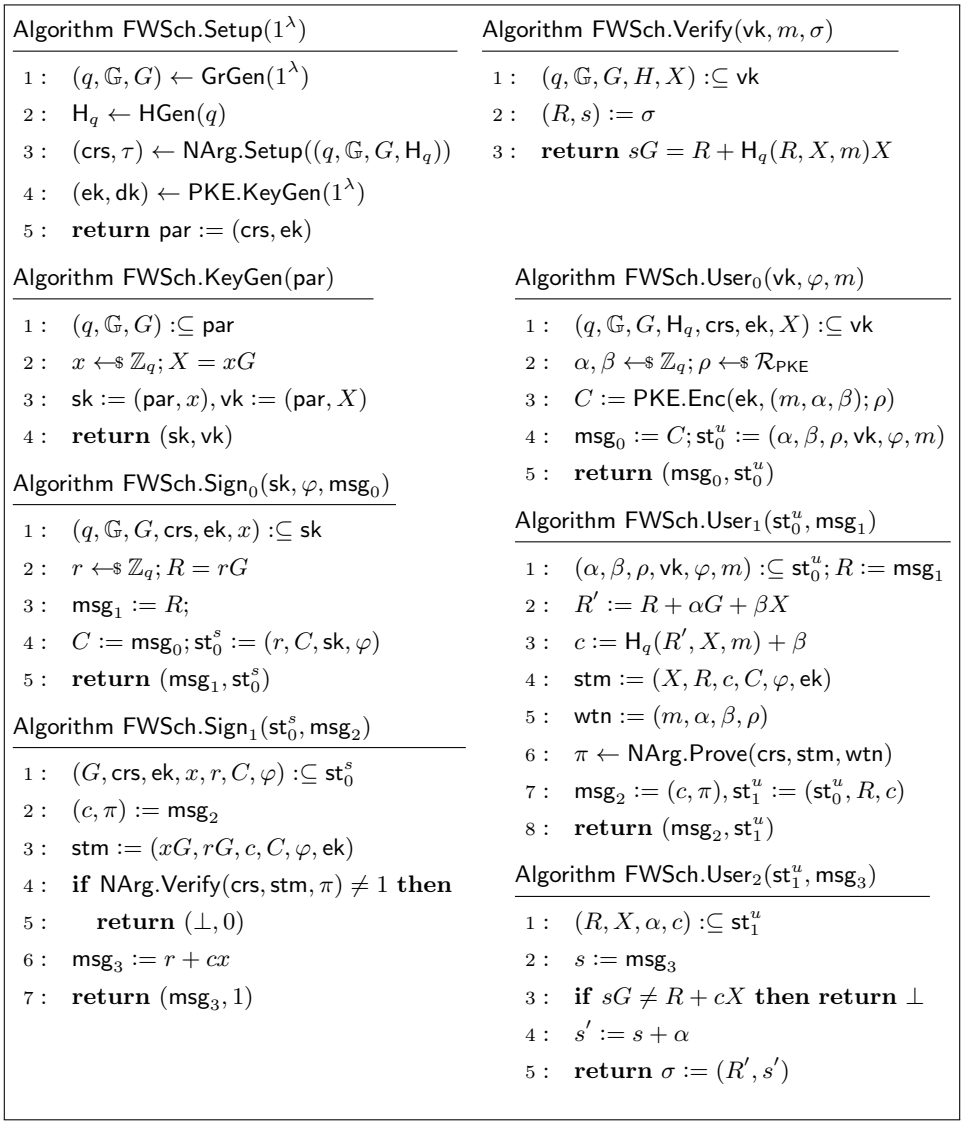


Fig. 16. FWSch[PKE, NArg[$\mathsf{R}_{\mathsf{FWSch}}$], P, GrGen, HGen] the predicate blind Schnorr signature from [FW24], based on predicate compiler P, a group generation algorithm GrGen, a hash generator HGen, a PKE scheme PKE and a NIZK argument system NArg for the relation $\mathsf{R}_{\mathsf{FWSch}}$ defined as: $\mathsf{R}_{\mathsf{FWSch}}((q, \mathbb{G}, G, \mathsf{H}_q), (X, R, c, C, \varphi, \mathsf{ek}), (m, \alpha, \beta, \rho)) = 1$ if and only if $c = \mathsf{H}_q(R + \alpha G + \beta X, X, m) + \beta \wedge \mathsf{P}(\varphi, m) = 1 \wedge \mathsf{PKE.Enc}(\mathsf{ek}, (m, \alpha, \beta); \rho) = C$.

scheme, decryption itself serves as the extractor. In the black-box setting of a generic straight-line extractable commitment Cmt, extraction can in principle fail. In that case, one may insert an intermediate hybrid $\mathrm{H}_{0/1}$ between H_0 and H_1 and show their indistinguishability; otherwise, the extractability property would be contradicted (because the extraction fails only with negligible probability, the games are indistinguishable). That is, we present the PKE-based instantiation in the main proof, the reduction for a general straight-line extractable commitment differs only by this standard hybrid argument.

To give a bit more detail, there are two equivalent ways to argue about the extractability of the commitment. (a) If we use the concrete extractable commitment described above (i.e., a Pedersen commitment augmented with a straight-line extractable WI argument obtained by applying Fischlin's transform to the associated Σ-protocol). Then extraction reduces to the extractability of the resulting non-interactive argument (see the discussion in App. A.6). (b) Alternatively, and what we prefer, we treat the straight-line extractable commitment as a black box: we run its extractor and record its output. The event of an incorrect extraction is handled when considering the adversary's NIWI and the validity of the proved relation, as we can argue about the soundness or the extractability of the commitment (see the hybrids below). In either presentation, the underlying reduction is the same; we adopt the PKE presentation only to simplify the exposition. Formally, we add such an advantage $\mathsf{Adv}^{\mathrm{SLE}}_{\mathsf{Cmt}}(\mathsf{J}, \lambda)$ when bound the ad-

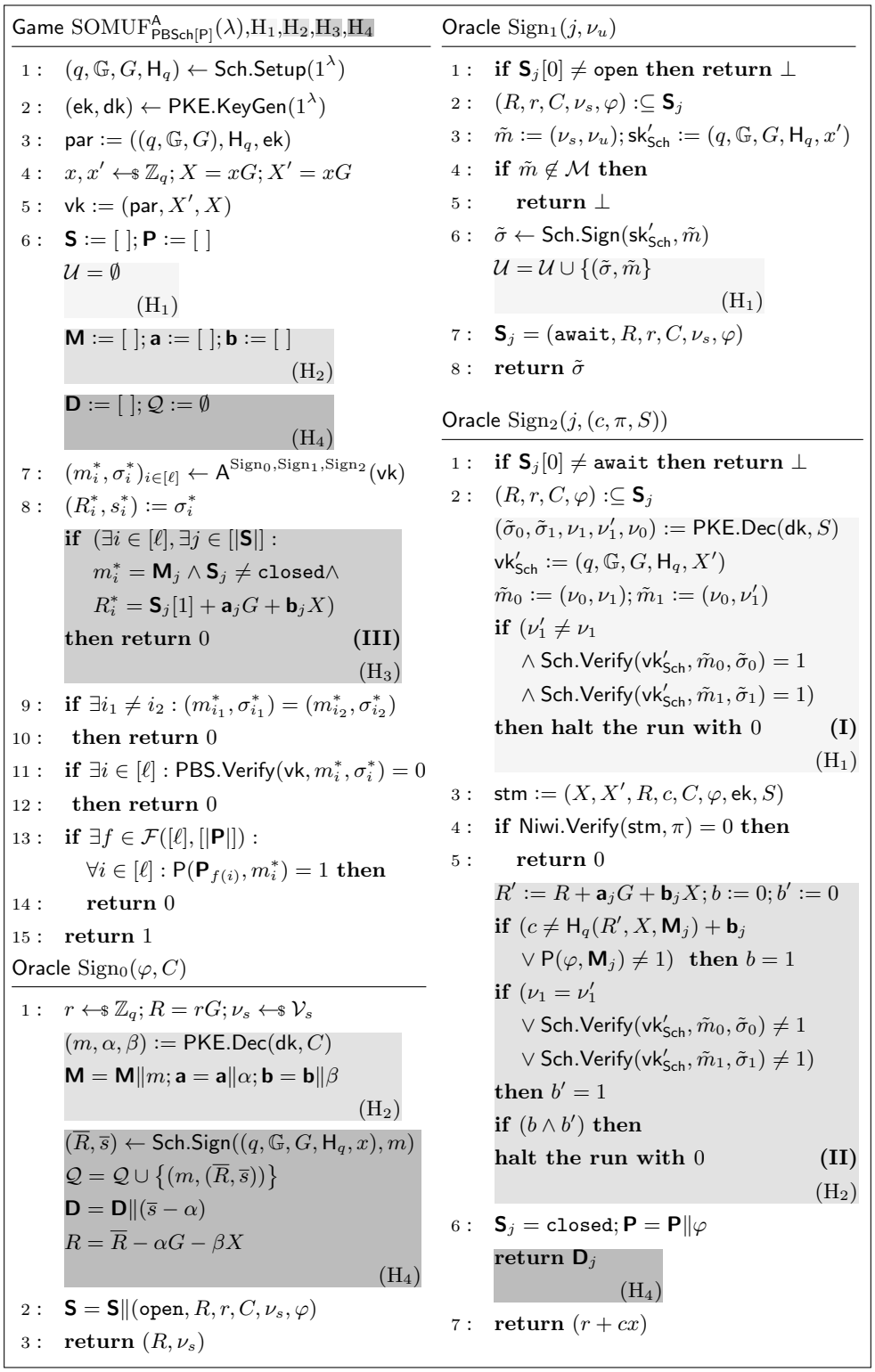


Fig. 17. The SOMUF game from Fig. 1 for $\mathsf{PBSch}[\mathsf{P}, \mathsf{GrGen}, \mathsf{HGen}, \mathsf{PKE}, \mathsf{Niwi}]$ from Fig. 3 and hybrid games used in the proof of Thm. 1. H_i includes all boxes with an index $\leq i$ and ignores all boxes with and index $> i$.

vantage of the SOMUF adversary. Where J is the adversary winning in the extractability game for the straight-line extractable commitment Cmt. According to Def. 17.

Reduction from unforgeability of Sch. We show that the difference between the advantage $\mathsf{Adv}^{\mathrm{H}_0}_{\mathsf{PBSch}}(\mathsf{A},\lambda)$ and the advantage $\mathsf{Adv}^{\mathrm{H}_1}_{\mathsf{PBSch}}(\mathsf{A},\lambda)$ is bounded by the advantage in winning the game SUF-CMA (Fig. 7) against the unforgeability of $\mathsf{Sch}[\mathsf{GrGen},\mathsf{HGen}]$ played by the adversary F, which returns a message-signature pair (that has not been previously issued by the signing oracle, thus a forgery) whenever the event E_1:

$$\mathsf{E}_1 : \Longleftrightarrow \ \nu_1 \neq \nu_1' \wedge \mathsf{Sch.Verify}(\mathsf{vk}'_{\mathsf{Sch}},(\nu_0,\nu_1),\tilde{\sigma}_0) = 1 \wedge \mathsf{Sch.Verify}(\mathsf{vk}'_{\mathsf{Sch}},(\nu_0,\nu_1'),\tilde{\sigma}_1) = 1$$

is true, if it happens then the game H_1 stops and returns 0 (as in Fig. 17). According to the definition of Strong Unforgeability (Def. 8), F receives as a challenge input a Schnorr verification key (sp, X') and has access to a signing oracle Sign that outputs signatures according to the signature key (sp, x'), such that $x'G = X'$, where G is a group generator specified in sp.

F, with sp, completes the $\mathsf{PBSch.Setup}$ by computing a key pair $(\mathsf{ek},\mathsf{dk})$ for PKE, then it also completes $\mathsf{Sch.KeyGen}$ by sampling another key $x \leftarrow\!\!\$\ \mathbb{Z}_q$, computing $X = xG$, and the verification key $\mathsf{vk} := (\mathsf{sp}, X', X)$ that A takes as input. Thus, F embeds its challenge Schnorr public key X' into the verification key for PBSch. Moreover, F initializes the set $\mathcal{U} := \emptyset$. The set $\mathcal{U}$ is used by F, when it simulates the oracle Sign_1 for A, to store the message-signature pairs issued by its signing oracle Sign. That is, F, when simulating Sign_1, asks the oracle Sign to sign the message requested by A during the Sign_1 execution, then updates the set $\mathcal{U}$ with the tuple composed of the signature output by the oracle and the message that Sign used to generate such a signature. Note that the corresponding secret key x' is not required since F, on each Sign_1 query by A, forwards the call to its signing oracle Sign.

F when it simulates the oracle Sign_2 for A, according to H_1, returns one of the two message-signature pairs $(\tilde{\sigma}_0,(\nu_0,\nu_1))$ or $(\tilde{\sigma}_0,(\nu_s,\nu_1'))$ whenever the event E_1 is reached. Specifically, F returns the first pair if $(\tilde{\sigma}_1,(\nu_0,\nu_1')) \in \mathcal{U}$, while it returns the second pair if $(\tilde{\sigma}_0,(\nu_0,\nu_1)) \in \mathcal{U}$. Thus, the adversary F wins the SUF-CMA game with the same probability that the event E_1 happens. That is, first, note that, when E_1 is true the fact that both message-signature pairs are valid, according to (sp, X'), comes from the perfect correctness of the PKE scheme and Sch scheme, namely because such message-signature pairs are decrypted from S according to the $\mathsf{PKE.Dec}$ algorithm and verified according to the $\mathsf{Sch.Verify}$ algorithm using the challenge Schnorr verification key (sp, X'). Second, the case that E_1 is reached and both the message-signature pairs are in $\mathcal{U}$ happens only if (in multiple sessions) the Sign oracle, called during the simulation of Sign_1 by F, signs two different messages $(a,b),(c,d) \in (\mathcal{V}_s \times \mathcal{V}_u) \subseteq \mathcal{M}$ with $a = c$ (i.e., with the same prefix), namely with the same probability that in two different oracle calls to Sign_0 the same ν_s (i.e., the prefix used in Sign_1) is sampled from $\mathcal{V}_s$. That is, looking at the behavior of Sign_1 and Sign_2 in Fig. 17, the fact that in $\mathcal{U}$ there are two message-signature pairs such that the two messages have the same prefix in $\mathcal{V}_s$ and different suffix in $\mathcal{V}_u$ occurs only if, in two different Sign_1 oracle calls, the same ν_s is sampled from $\mathcal{V}_s$ twice and added to two different $\mathbf{S}_j$ and $\mathbf{S}_{j'}$ with $j \neq j'$. Since $\mathcal{V}_s$ has super-polynomial cardinality in the security parameter, the probability of sampling the same element twice is $1/|\mathcal{V}_s|$, which is negligible. More precisely, by a simple birthday-bound argument, over q oracle calls, this happens with probability at most $q^2/|\mathcal{V}_s|$, which is negligible. Thus, the fact that exactly one of the two message-signature pairs, (ν_0,ν_1) or (ν_0,ν_1'), belongs to $\mathcal{U}$ occurs with overwhelming probability. Consequently, the pair not in $\mathcal{U}$ constitutes a valid forgery for F.

We conclude that F wins the SUF-CMA game whenever E_1 holds in H_1, and therefore:

$$\mathsf{Adv}^{\mathrm{SOMUF}}_{\mathsf{PBSch}}(\mathsf{A},\lambda) \leq \mathsf{Adv}^{\mathrm{SUF\text{-}CMA}}_{\mathsf{Sch}[\mathsf{GrGen},\mathsf{HGen}]}(\mathsf{F},\lambda) + \frac{q^2}{|\mathcal{V}_s|} + \Pr[\mathrm{H}_1(\mathsf{A},\lambda) = 1] \tag{2}$$

Game H_2. In H_2 we introduce three lists $\mathbf{M}, \mathbf{a}$ and $\mathbf{b}$ and modify the oracle Sign_0, so that on each call with input (φ, C) we decrypt C to obtain the values (m,α,β).

Note that, as mentioned earlier, due to the perfect correctness of the PKE, we can perform decryption in the game without introducing an additional hybrid game. Following what was said before, however, we could readily construct an intermediate game $\mathrm{H}_{1/2}$ between H_1 and H_2 to show that these games are indistinguishable (otherwise, this would break the extractability property of the commitment scheme). However, our presentation as a PKE simplifies the proof without loss of generality.

Formally, we also add such an advantage $\mathsf{Adv}^{\mathrm{SLE}}_{\mathsf{Cmt}}(\mathsf{K},\lambda)$ when bound the advantage of the SOMUF adversary. Where K is the adversary winning in the extractability game for the straight-line extractable commitment Cmt. According to Def. 17.

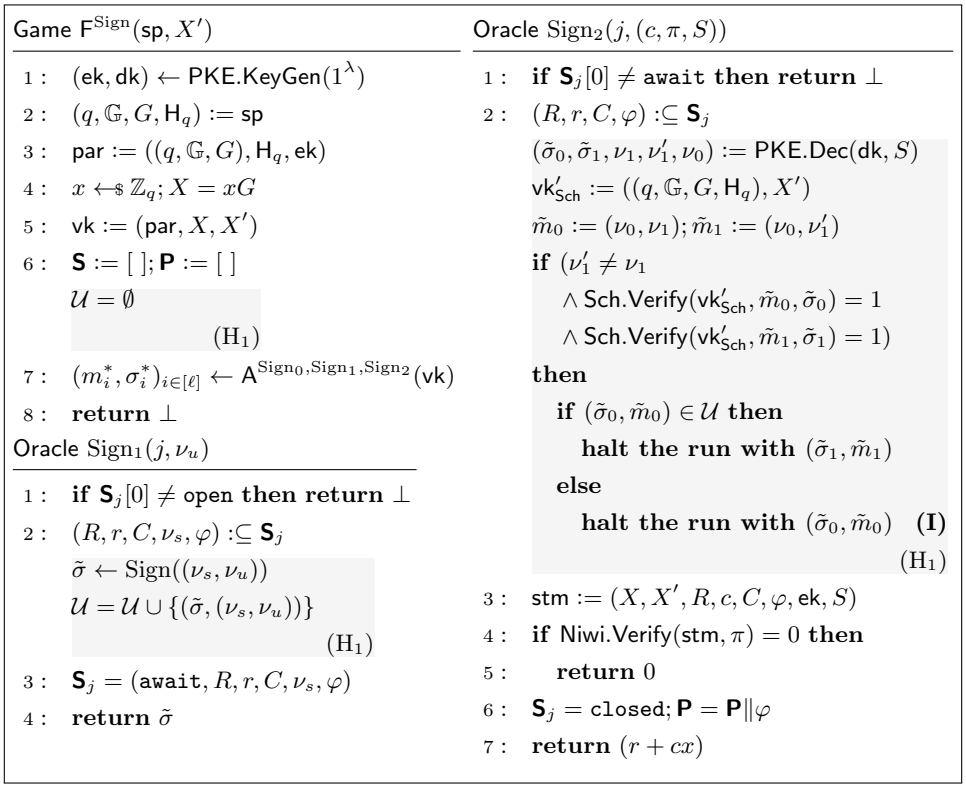


Fig. 18. F playing against SUF-CMA security of $\mathsf{Sch}[\mathsf{GrGen}, \mathsf{HGen}]$. The oracle Sign_0 is simulated to A as defined in game H_1 in Fig. 17.

After decryption, we then append to the lists $\mathbf{M} = \mathbf{M} \| m, \mathbf{b} = \mathbf{b} \| \beta$ and $\mathbf{a} = \mathbf{a} \| \alpha$. In each Sign_2 call on input $(j, (c, \pi, S))$, we check if the events $\mathsf{E}_{2.1}$ and $\mathsf{E}_{2.2}$ are both true, if so we stop the game and return 0. Where the event $\mathsf{E}_{2.1}$, for the decrypted values at index j and $R' := R + \mathbf{a}_j G + \mathbf{b}_j X$ is defined as follows:

$$\mathsf{E}_{2.1} :\iff c \neq \mathsf{H}_q(R', X, \mathbf{M}_j) + \mathbf{b}_j \vee \mathsf{P}(\varphi, \mathbf{M}_j) \neq 1$$

Concretely the event $\mathsf{E}_{2.1}$ states that the c given as input to Sign_2 is not equal to the value $\mathsf{H}_q(R', X, \mathbf{M}_j) + \mathbf{b}_j$ or that under the predicate complier P the message $\mathbf{M}_j$ is not valid for the predicate φ, that is in $\mathbf{S}_j$. The event $\mathsf{E}_{2.2}$ for $\mathsf{vk}'_{\mathsf{Sch}} := ((q, \mathbb{G}, G, \mathsf{H}_q), X')$, is defined as follows:

$$\mathsf{E}_{2.2} :\iff \nu_1 = \nu_1' \vee \mathsf{Sch.Verify}(\mathsf{vk}'_{\mathsf{Sch}}, (\nu_s, \nu_1), \tilde{\sigma}_0) \neq 1 \vee \mathsf{Sch.Verify}(\mathsf{vk}'_{\mathsf{Sch}}, (\nu_s, \nu_1'), \tilde{\sigma}_1) \neq 1$$

Concretely, the event $\mathsf{E}_{2.2}$, given the tuple $(\tilde{\sigma}_0, \tilde{\sigma}_1, \nu_1, \nu_1', \nu_0)$ that is the message decrypted from the ciphertext S, states that the two values (ν_1, ν_1') are equal, or that the signature $\tilde{\sigma}_0$ is not a valid signature for the message $\tilde{m}_0 := (\nu_0, \nu_1)$ using $\mathsf{vk}'_{\mathsf{Sch}}$ as the verification key, or that the signature $\tilde{\sigma}_1$ is not a valid signature for the message $\tilde{m}_1 := (\nu_0, \nu_1')$ using $\mathsf{vk}'_{\mathsf{Sch}}$ as the verification key.

Reduction from soundness of Niwi**.** We show that the difference between the advantage $\mathsf{Adv}^{\mathrm{H}_1}_{\mathsf{PBSch}}(\mathsf{A}, \lambda)$ and the advantage $\mathsf{Adv}^{\mathrm{H}_2}_{\mathsf{PBSch}}(\mathsf{A}, \lambda)$ is bounded by the advantage of winning the SND game (Def. 10) against the soundness of $\mathsf{Niwi}[\mathsf{R}_{\mathsf{PBSch}}]$, played by the adversary S who returns the statement-proof pair whenever H_2 aborts (as defined in Fig. 19). According to the definition of the SND game for Niwi over the relation $\mathsf{R}_{\mathsf{PBSch}}$, S receives as input $\mathsf{par}_{\mathsf{R}}$ generated by $\mathsf{Niwi.Rel}$, which is defined as $\mathsf{Sch.Setup}$. The adversary S in the SND game thus perfectly simulates H_2 for the adversary A until an abort occurs. In Sign_2, S checks whether, for a valid statement-proof pair (stm, π), parts of the supposed witness $(m, \alpha, \beta, \tilde{\sigma}_0, \tilde{\sigma}_1, \nu_1, \nu_1', \nu_0)$ satisfy $\mathsf{E}_{2.1}$ and $\mathsf{E}_{2.2}$ (i.e., if (m, α, β) satisfy $\mathsf{E}_{2.1}$ and $(\tilde{\sigma}_0, \tilde{\sigma}_1, \nu_1, \nu_1', \nu_0)$ satisfy $\mathsf{E}_{2.2}$). If so, S returns the pair (stm, π). Thus, S returns a pair (stm, π) if and only if H_2 aborts at line **(II)**.

It remains to show that when this happens, S wins the SND game.

First, observe that the negation of the event E_1 is precisely the event $\mathsf{E}_{2.2}$. This follows directly from an application of De Morgan's law to the condition defining the event E_1. Keeping in mind that $\neg \mathsf{E1} = \mathsf{E}_{2.2}$, assume S reaches line **(II)** (i.e., no abort occurs at line **(I)**) during a call to Sign_2 on input

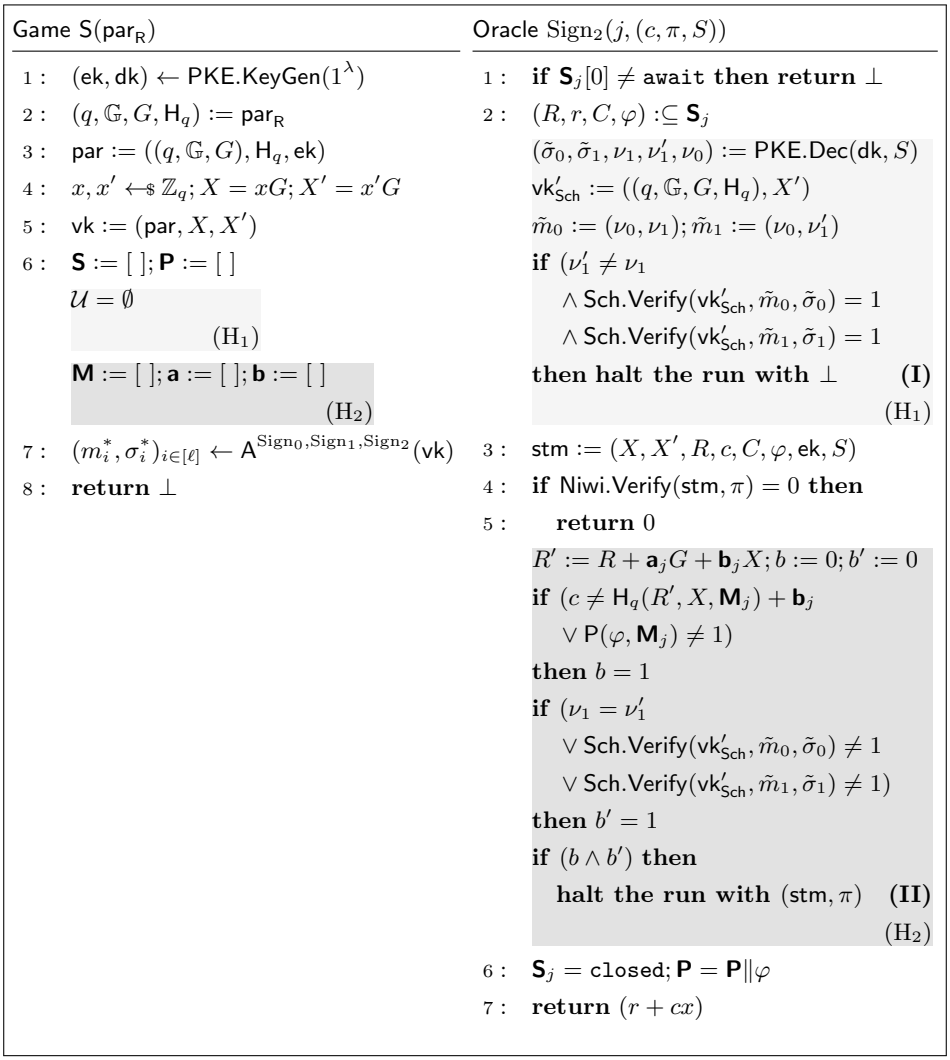


Fig. 19. S playing against SND security of $\mathsf{Niwi}[\mathsf{R}_{\mathsf{PBSch}}]$. The oracles Sign_0 and Sign_1 are simulated to A as defined in game H_2 in Fig. 17.

$(j,(c,\pi,S))$. Let $(\mathsf{stm} := (X,X',R,c,C,\varphi,\mathsf{ek},S),\pi)$ be its output. The condition at line **(II)** is reached only if π is an accepting proof for the statement stm.

Thus, it suffices to show that stm is not a valid statement. Towards a contradiction, assume stm is a valid statement. Since we have reached the line **(II)**, the event $\neg\mathsf{E}_1$ must have occurred; otherwise, there would have been an abort at line **(I)**. Consequently, given that $\neg\mathsf{E}_1 = \mathsf{E}_{2.2}$, we have $b' = 1$. This rules out the existence of a part of the supposed witness $(\tilde\sigma_0^\star,\tilde\sigma_1^\star,\nu_1^\star,\nu_1'^\star,\nu_0^\star,\varrho^\star)$ satisfying:

$$\nu_1^\star \neq \nu_1'^\star \wedge S = \mathsf{PKE.Enc}(\mathsf{ek},(\tilde\sigma_0^\star,\tilde\sigma_1^\star,\nu_1^\star,\nu_1'^\star,\nu_0^\star);\varrho^\star) \wedge$$
$$\mathsf{Sch.Verify}((\mathsf{sp},X'),(\nu_0^\star,\nu_1^\star),\tilde\sigma_0^\star) = 1 \wedge \mathsf{Sch.Verify}((\mathsf{sp},X'),(\nu_0^\star,\nu_1'^\star),\tilde\sigma_1^\star) = 1$$

and thus, enabling $\mathsf{R}_{\mathsf{PBSch}} = 1$. Otherwise, there would have been an abort at line **(I)**.

Thus, we are left with the case that there must exist a part of the supposed witness $(m^\star,\alpha^\star,\beta^\star,\rho^\star)$ enabling $\mathsf{R}_{\mathsf{PBSch}}$ to be equal to 1, because:

$$c = \mathsf{H}_q(R',X,m^\star) + \beta^\star \wedge \mathsf{P}(\varphi,m^\star) = 1 \wedge C = \mathsf{PKE.Enc}(\mathsf{ek},(m^\star,\alpha^\star,\beta^\star;\rho^\star)) \tag{3}$$

where $R' := R + \alpha^\star G + \beta^\star X$. One again, if stm is a valid statement, there must exist a part of the supposed witness $(m^\star,\alpha^\star,\beta^\star,\rho^\star)$. By the definition of S, we have $(m,\beta,\alpha) = \mathsf{PKE.Dec}(\mathsf{dk},C)$. By the perfect correctness of PKE, together with the first clause in Equation (3), this can only be the case if $m = m^\star$, $\alpha = \alpha^\star$, and $\beta = \beta^\star$. However, by the definition of S (since we assume it reaches line **(II)**), we know that $\mathsf{E}_{2.1}$ is true, implying $c \neq \mathsf{H}_q(R',X,m) + \beta \vee \mathsf{P}(\varphi,m) \neq 1$. This contradicts $m = m^\star$, $\alpha = \alpha^\star$, and $\beta = \beta^\star$, as well as Equation (3). Therefore, such a witness does not exist (i.e., when both b and b'

are equal to 1, the witness cannot exist). This means that whenever line **(II)** is reached in H_2, S wins the SND game against $\mathsf{Niwi}[\mathsf{RPBSch}]$. Consequently:

$$\Pr[\mathrm{H}_1(\mathsf{A},\lambda)=1] \leq \mathsf{Adv}^{\mathrm{SND}}_{\mathsf{Niwi}[\mathsf{R}_{\mathsf{PBSch}}]}(\mathsf{A},\lambda) + \Pr[\mathrm{H}_2(\mathsf{A},\lambda)=1] \tag{4}$$

Game H_3**.** In H_3, we define the event E_3 as follows:

$$\mathsf{E}_3 \iff \exists i \in [\ell], \exists j \in [|\mathbf{S}|] : m_i^* = \mathbf{M}_j \wedge \mathbf{S}_j \neq \texttt{closed} \wedge R_i^* = \tilde{R}_j + \mathbf{a}_j G + \mathbf{b}_j X$$

where $\tilde{R}_j := \mathbf{S}_j[1]$. The event E_3 is checked after the adversary produces its final output. If E_3 holds, the game H_3 returns 0, as described in Fig. 17.

Concretely, the event E_3 indicates that for at least one of the messages m_i^* in the adversary's final output, there exists a session j where: the message m_i^* was decrypted during an Sign_0 oracle call (i.e., $\mathbf{M}_j = m_i^*$); the session j was not successfully closed via a call to the Sign_2 oracle (i.e., $\mathbf{S}_j \neq \texttt{closed}$); the first part of the message's Schnorr signature, R_i^*, satisfies the relationship $R_i^* = \tilde{R}_j + \mathbf{a}_j G + \mathbf{b}_j X$, where $\tilde{R}_j := \mathbf{S}_j[1]$ corresponds to the output of the j-th Sign_0 oracle call. Here, $\mathbf{a}_j$ and $\mathbf{b}_j$ were derived during decryption in the j-th session.

We observe the following relationship, $\Pr\big[\mathrm{H}_3^{\mathsf{A}}(\lambda)=1\big] = \Pr\big[\mathrm{H}_2^{\mathsf{A}}(\lambda)=1 \wedge \neg\mathsf{E}_3\big]$, since H_2 and H_3 are identical unless E_3 occurs (i.e., when line 9 is reached in H_3). Using the law of total probability, we further decompose $\Pr\big[\mathrm{H}_2^{\mathsf{A}}(\lambda)=1\big]$ as $\Pr\big[\mathrm{H}_2^{\mathsf{A}}(\lambda)=1\big] = \Pr\big[\mathrm{H}_2^{\mathsf{A}}(\lambda)=1 \wedge \mathsf{E}_3\big] + \Pr\big[\mathrm{H}_2^{\mathsf{A}}(\lambda)=1 \wedge \neg\mathsf{E}_3\big]$. Combining these equations, we derive:

$$\Pr\big[\mathrm{H}_2^{\mathsf{A}}(\lambda)=1\big] = \Pr\big[\mathrm{H}_2^{\mathsf{A}}(\lambda)=1 \wedge \mathsf{E}_3\big] + \Pr\big[\mathrm{H}_3^{\mathsf{A}}(\lambda)=1\big] \tag{5}$$

Reduction to the hardness of DL. We show that the difference between the advantage $\mathsf{Adv}^{\mathrm{H}_2}_{\mathsf{PBSch}}(\mathsf{A},\lambda)$ and the advantage $\mathsf{Adv}^{\mathrm{H}_3}_{\mathsf{PBSch}}(\mathsf{A},\lambda)$ is bounded by the advantage of winning the DLOG game (Fig. 5), and thus breaking the assumed hardness of the DL problem, played by the adversary D who returns the solution of a discrete log whenever H_3 returns 0 in line **(III)**.

We bound $\Pr\big[\mathrm{H}_2^{\mathsf{A}}(\lambda)=1 \wedge \mathsf{E}_3\big]$ by the advantage against the discrete-logarithm (DL) hardness of GrGen of an algorithm D. The reduction proceeds in two steps. First, we provide a reduction to wOMDL via the adversary L given in Fig. 20; then we apply Lemma 1 to reduce to the hardness of DL. By the definition of wOMDL game , L receives as input the group parameters $(q,\mathbb{G},G)$. With that it chooses a hash function $\mathsf{H}_q \leftarrow \mathsf{HGen}(q)$ and then it completes the $\mathsf{PBSch.Setup}$ and $\mathsf{PBSch.KeyGen}$ by computing the verification key vk (very similarly to S). Then, L simulates H_2 for A, where in each oracle call of Sign_0, L queries its challenge oracle $R \leftarrow \mathrm{Chal}()$. Since the oracle returns uniformly sampled elements, the simulation is perfect up to this point. If A closes a session with session number j successfully with a call to Sign_2, L obtains $r := \mathrm{DLog}(j)$ from its DLog oracle. Assume A satisfies the condition expressed by the event E_3. Thus, some session j, with challenge $R_j := \mathbf{S}_j[1]$, was not closed (i.e., $\mathbf{S}_j \neq \texttt{closed}$), and so the oracle $r_j := \mathrm{DLog}(j)$ was not called either, and for some index $i \in [\ell]$, we have $R_j + \mathbf{a}_j G + \mathbf{b}_j X = R_i^*$. When A wins H_2, we know by the validity of the signature that $s_i^* G = R_i^* + \mathsf{H}_q(R_i^*, X, m_i^*)X$ (i.e., since condition at line 11 of game H_2 is not fulfilled).

By combining these two equations, we obtain that $s_i^* G = r_j G + \mathbf{a}_j G + \mathbf{b}_j x G + \mathsf{H}_q(R_i^*, X, m_i^*)xG$, and thus that s_i^* is exactly $r_j + \mathbf{a}_j + \mathbf{b}_j x + \mathsf{H}_q(R_i^*, X, m_i^*)x$. From this, L computes and returns $rj := \log_G(R_j)$ and thereby wins wOMDL game (r_j is the dlog of a challenge that was not solved by the DLog oracle). Therefore, for now we obtain that $\Pr\big[\mathrm{H}_2^{\mathsf{A}}(\lambda)=1 \wedge \mathsf{E}_3\big] \leq \mathsf{Adv}^{\mathrm{wOMDL}}_{\mathsf{GrGen}}(\mathsf{L},\lambda)$. Moreover, let q be an upper-bound of successfully closed sessions via queries to the Sign_2 oracle made by A. Then L's number of queries to its DLog oracle is also bounded by q, and by applying Lemma 1 we obtain an adversary D playing in the game DLOG where $\Pr\big[\mathrm{H}_2^{\mathsf{A}}(\lambda)=1 \wedge \mathsf{E}_3\big] \leq \mathsf{Adv}^{\mathrm{DLOG}}_{\mathsf{GrGen}}(\mathsf{D},\lambda)$. Finally, by (5) we obtain that:

$$\Pr\big[\mathrm{H}_2^{\mathsf{A}}(\lambda)=1\big] \leq \mathsf{Adv}^{\mathrm{DLOG}}_{\mathsf{GrGen}}(\mathsf{D},\lambda) + \Pr\big[\mathrm{H}_3^{\mathsf{A}}(\lambda)=1\big] \tag{6}$$

Game H_4**.** In the game H_4 we introduce a list $\mathbf{D}$ and a set $\mathcal{Q}$, and we modify Sign_0 so that after decrypting C into (m,α,β), we compute a Schnorr signature on m under the signing key $\mathsf{sk}_{\mathsf{Sch}} := (q,\mathbb{G},G,\mathsf{H}_q,x)$[19] by running $(\overline{R},\overline{s}) \leftarrow \mathsf{Sch.Sign}(\mathsf{sk}_{\mathsf{Sch}}, m)$, then we replace the random signer challenge $R := rG$ by $R := \overline{R} - \alpha G - \beta X$. Note that $\mathsf{Sch.Sign}$ returns a uniform $\overline{R}$ and therefore R is a uniform element. The simulation is perfect up to here. Moreover, in Sign_0, we also compute and append to the list $\mathbf{D}$ the value $(\overline{s}-\alpha)$,

[19] Note that before in game H_1, according to Fig. 17, we have defined $\mathsf{vk}'_{\mathsf{Sch}}$ that is the verification key for $\mathsf{sk}'_{\mathsf{Sch}} := (q,\mathbb{G},G,\mathsf{H}_q,x')$.

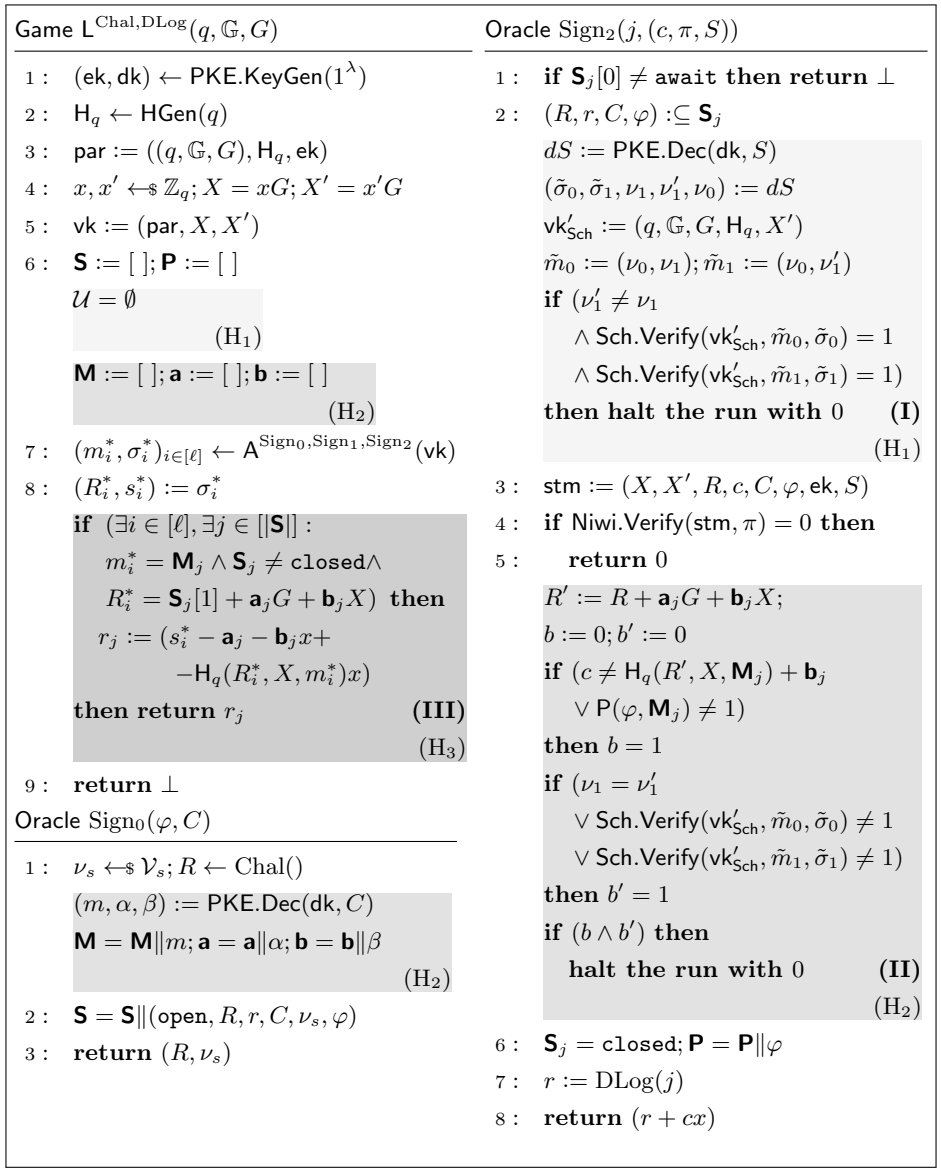


Fig. 20. L playing against wOMDL security. The oracles Sign_0 and Sign_1 are simulated to A as defined in game H_3 in Fig. 17.

and we also add to the set $\mathcal{Q}$ the value $(m, (\overline{R}, \overline{s}))$. Finally, we also modify Sign_2 and instead of returning $s := (r+cx)$ we return the value we previously stored in $\mathbf{D}$ that is $s := \mathbf{D}_j = (\overline{s}-\alpha)$. The user playing game H_4 after interacting with Sign_2 now obtains simulated elements $(R, s) = (\overline{R}-\alpha G-\beta X, \overline{s}-\alpha)$, because of the $\mathsf{Sch.Sign}$ algorithm and since line 6 was reached (i.e., no abort happens in lines **(I)** and **(II)**), we have $c = \mathsf{H}_q(R + \alpha G + \beta X, X, m) + \beta$. For any choice of α, β and m, and the signing key sk, the user's view in H_4 is distributed equivalently to its view in H_3. The view in H_3 is defined as $\{(R, \nu_s, \tilde\sigma, s) | r \leftarrow\!\$\ \mathbb{Z}_q; R = rG; \nu_s \leftarrow\!\$\ \mathcal{V}_s; \tilde\sigma \leftarrow \mathsf{Sch.Sign}(\mathsf{sk}'_{\mathsf{Sch}}, (\nu_s, \nu_u)); s = r + \mathsf{H}_q(R + \alpha G + \beta X, X, m)x + \beta x\}$ that is equal to $\{(\overline{R} - \alpha G - \beta X, \nu_s, \tilde\sigma, s) | \overline{r} \leftarrow\!\$\ \mathbb{Z}_q; \overline{R} = \overline{r}G; \nu_s \leftarrow\!\$\ \mathcal{V}_s; \tilde\sigma \leftarrow \mathsf{Sch.Sign}(\mathsf{sk}'_{\mathsf{Sch}}, (\nu_s, \nu_u)); s = \overline{r} - \alpha - \beta x + \mathsf{H}_q(\overline{R}, X, m)x + \beta x\}$. Since $\overline{r} - \alpha - \beta x$ is distributed as r; when we set $s = \overline{s} - \alpha$ we obtain an equal to the previous distribution, that is $\{(\overline{R} - \alpha G - \beta X, \nu_s, \tilde\sigma, \overline{s} - \alpha) | \overline{r} \leftarrow\!\$\ \mathbb{Z}_q; \overline{R} = \overline{r}G; \nu_s \leftarrow\!\$\ \mathcal{V}_s; \tilde\sigma \leftarrow \mathsf{Sch.Sign}(\mathsf{sk}'_{\mathsf{Sch}}, (\nu_s, \nu_u)); \overline{s} = \overline{r} + \mathsf{H}_q(\overline{R}, X, m)x\}$ that is equal to $\{(\overline{R} - \alpha G - \beta X, \nu_s, \tilde\sigma, \overline{s} - \alpha) | \nu_s \leftarrow\!\$\ \mathcal{V}_s; \tilde\sigma \leftarrow \mathsf{Sch.Sign}(\mathsf{sk}'_{\mathsf{Sch}}, (\nu_s, \nu_u)); (\overline{R}, \overline{s}) \leftarrow \mathsf{Sch.Sign}(\mathsf{sk}_{\mathsf{Sch}}, m)\}$, which is precisely the view in H_4. Thus, we obtain that:

$$\Pr\big[\mathrm{H}_3^{\mathsf{A}}(\lambda) = 1\big] = \Pr\big[\mathrm{H}_4^{\mathsf{A}}(\lambda) = 1\big] \tag{7}$$

Reduction from unforgeability of Sch. To finish the proof, we construct an adversary C in Fig. 21 that succeeds in the SUF-CMA game against the Schnorr signature scheme $\mathsf{Sch}[\mathsf{GrGen}, \mathsf{HGen}]$ with probability equal to $\Pr\big[\mathrm{H}_4^{\mathsf{A}}(\lambda) = 1\big]$. By the definition of SUF-CMA, C receives as challenge input a Schnorr

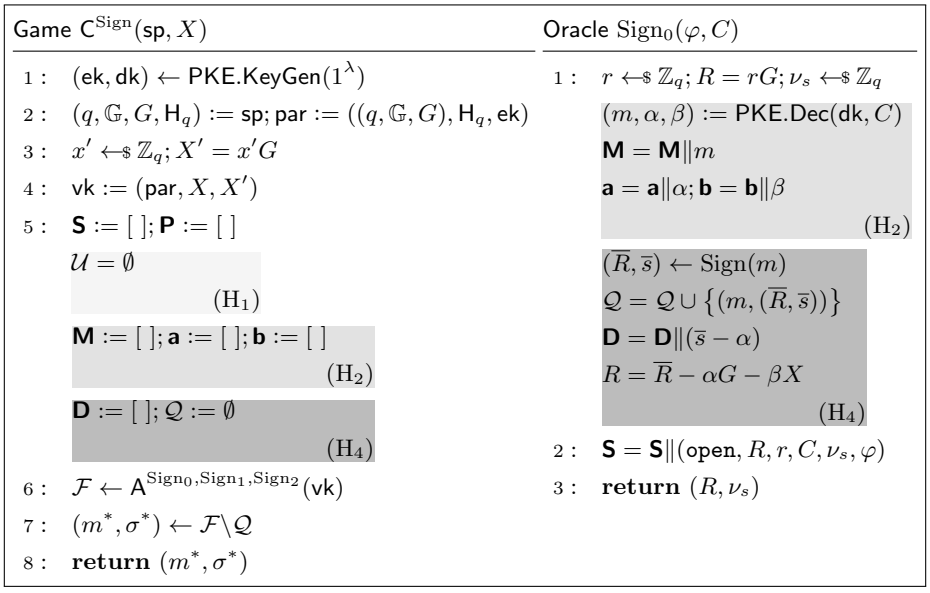


Fig. 21. C playing against SUF-CMA security of Sch[GrGen, HGen]. The oracles Sign_1 and Sign_2 are simulated to A as defined in game H_4 in Fig. 17.

verification key that is $(\mathsf{sp}, X) := \mathsf{vk}_{\mathsf{Sch}}$ and has access to a signing oracle Sign, such oracle issue signature that verifies with $\mathsf{vk}_{\mathsf{Sch}}$. Note that in the reduction for the game H_1 we also reduce to the SUF-CMA game against the Schnorr signature scheme, but the adversary F had $(\mathsf{sp}, X') := \mathsf{vk}'_{\mathsf{Sch}}$ as verification key. With the Schnorr parameters sp, the adversary C completes PBSch.Setup and PBSch.KeyGen the very same way F does in Fig. 18, moreover C initializes a empty set $\mathcal{Q}$ used to remember the message-signature pairs issued from its signing oracle Sign. When C simulates H_3 for A, it embeds its challenge Schnorr public key X into the verification key for PBSch. The corresponding secret key is not required since C on each Sign_0 query by A forwards the call to its signing oracle Sign. The simulation is perfect.

We show that if A wins H_4 outputting $\mathcal{F} = \{(m_i^*, \sigma_i^*)\}_{i \in [\ell]}$, then this set must contain a successful forgery that can be used by C, that is, an element that is not contained in $\mathcal{Q} = \left\{(m_j, \sigma_j := (\overline{R}_j, \overline{s}_j))\right\}_{j \in [|\mathbf{S}|]}$ (where index j corresponds to the signing session number in which the pair was added to $\mathcal{Q}$). Letting Γ be the set of indices of the sessions that were eventually closed, we can define $\mathcal{Q}_c := \{(m_j, \sigma_j)\}_{j \in \Gamma}$. Consequently, we define $\mathcal{Q}_o := \mathcal{Q} \backslash \mathcal{Q}_c$.

First, we show that if A wins H_4 there exists an element $(m_{i^*}^*, \sigma_{i^*}^*) \in \mathcal{F}$ that is not in $\mathcal{Q}_c$. If we had $\mathcal{F} \subseteq \mathcal{Q}_c$, then there would exist an injective function $f : [n] \to \Gamma$ mapping elements of $\mathcal{F}$ to elements of $\mathcal{Q}_c$ (i.e., $m_i^* = m_{f(i)}$). For all $j \in \Gamma$ (the closed sessions), we have $\mathsf{P}(\mathbf{P}_j, m_j) = 1$, as otherwise H_4 would have aborted in line **(II)**[20]. We thus have $1 = \mathsf{P}(\mathbf{P}_{f(i)}, m_{f(i)}) = \mathsf{P}(\mathbf{P}_{f(i)}, m_i^*)$ for all $i \in [\ell]$, which contradicts the winning condition of H_4, which requires that no such f exists.

We next show that $(m_{i^*}^*, \sigma_{i^*}^*) \notin \mathcal{Q}_o$, namely, that such message-signature pair was not obtained in an unfinished session either. Towards a contradiction, assume for some $j \notin \Gamma$ we have $(m_{i^*}^*, (R_{i^*}^*, s_{i^*}^*)) = (m_j, (\overline{R}_j, \overline{s}_j))$.

Then we would have (1) $m_i^* = m_j = \mathbf{M}_j$, since $\mathbf{M}$ contains the same messages that are in the set $\mathcal{Q}$, (2) $\mathbf{S}_j \neq \texttt{closed}$, since the session was not closed, and considering the value R in the definition of Sign_0, we have $\mathbf{S}_j[1] = \overline{R}_j - \alpha_j^* G - \beta_j^* X$, which together with $\overline{R}_j = R_{i^*}^*$ yields (3) $R_{i^*}^* = \mathbf{S}_j[1] + \mathbf{a}_j G + \mathbf{b}_j X$. Now the existence of values i^* and j, considering the point in (1) and (3), leads precisely to an abort of H_4 in line **(III)** (i.e., we describe exactly the fulfillment of the event E_3).

We have thus shown that $(m_{i^*}^*, \sigma_{i^*}^*)$ is neither in $\mathcal{Q}_c$, nor in $\mathcal{Q}_o$. Since $\mathcal{Q}_o := \mathcal{Q} \backslash \mathcal{Q}_c$, it follows that $(m_{i^*}^*, \sigma_{i^*}^*)$ is not in $\mathcal{Q}$. Consequently, this constitutes a valid forgery for C. Therefore, we have:

$$\Pr\left[\mathrm{H}_4^{\mathsf{A}}(\lambda) = 1\right] \leq \mathsf{Adv}^{\text{SUF-CMA}}_{\mathsf{Sch}[\mathsf{GrGen}, \mathsf{HGen}]}(\mathsf{C}, \lambda) \tag{8}$$

The proof now follows from Equations (2,4,6,7,8). □

[20] Remember that in H_3 when we arrive at line **(II)** we always have that $b' = 1$, otherwise H_3 has already aborted in line **(I)**.

C Proof of Theorem 2

We give a formal proof that our predicate blind signature scheme PBSch from Fig. 3 satisfies blindness as defined in Def. 3. The proof works via reductions to the security of the underlying building blocks, namely, the WI property of Niwi and the CPA-security of PKE. As noted in App. B, we use the latter merely for convenience and to enhance readability; it can be smoothly substituted without any loss of details with the hiding property of a straight-line extractable commitment scheme (i.e., using such a tool in a black-box way instead of explicitly employing the PKE scheme).

We proceed by showing a sequence of games specified in Fig. 22. We recall that in the blindness game according to Fig. 2 the adversary is defined as $\mathsf{A} = (\mathsf{A}_1, \mathsf{A}_2)$, thus in the following section we directly refer to A_1 and A_2 when necessary. In this section, we employ the keyword "**halt the run with**" inside an oracle to explicitly denote the termination of the game and its output, rather than the return value of the oracle.

For clarity, we begin by outlining the structure of the proof. To simplify the exposition, we first present the theorem in the case where the commitment Cmt is instantiated via a PKE scheme PKE, where the setup safely generates a key pair. This case is particularly convenient, as it lets us focus on the computation of the NIWI argument, which is the core part of our construction. We remark that when using another straight-line extractable commitment (e.g., the NPRO-based instantiation discussed above), there is no difference: such tools can be employed in a black-box manner.

Game H_0**.** This is the BLD game from Fig. 2, where PBS is instantiated with PBSch as defined in Fig. 3. Specifically, the algorithms $\mathsf{PBS.Setup}$, $\mathsf{PBS.KeyGen}$, $\mathsf{PBS.User}_0$, $\mathsf{PBS.User}_1$, $\mathsf{PBS.User}_2$, and $\mathsf{PBS.User}_3$ are replaced with their respective instantiations from Fig. 3. Thus, we have that $\mathsf{Adv}^{\mathrm{BLD}}_{\mathsf{PBSch}}(\mathsf{A}, \lambda) = \mathsf{Adv}^{\mathrm{H}_0}(\mathsf{A}, \lambda)$ where:

$$\mathsf{Adv}^{\mathrm{H}_0}(\mathsf{A}, \lambda) := \left| \Pr\left[{\mathrm{H}_0}^{\mathsf{A},1}(\lambda) = 1 \right] - \Pr\left[{\mathrm{H}_0}^{\mathsf{A},0}(\lambda) = 1 \right] \right|$$

Game H_{0-1}**.** In H_{0-1}, we introduce two variables: rwd, initialized to 0, and rwdmsg, initialized to $\perp$. Additionally, we modify the behavior of the User_2 oracle. Specifically, during a call to User_2, H_{0-1} assigns the variable rwdmsg the tuple $(\sigma_0^\star, \sigma_1^\star, \nu_{01}^\star, \nu_{11}^\star, \nu_0^\star)$, which is defined as follows.

Given the verification key $\mathsf{vk}'_{\mathsf{Sch}} := ((q, \mathbb{G}, G, \mathsf{H}_q), X')$, we have $\mathsf{Sch.Verify}(\mathsf{vk}'_{\mathsf{Sch}}, (\nu_0^\star, \nu_{01}^\star), \sigma_0^\star) = 1$ and $\mathsf{Sch.Verify}(\mathsf{vk}'_{\mathsf{Sch}}, (\nu_0^\star, \nu_{11}^\star), \sigma_1^\star) = 1$. That is, during the oracle call to User_2, the variable rwdmsg is assigned a tuple containing two valid signatures $(\sigma_0^\star, \sigma_1^\star)$ and two messages $(\nu_{01}^\star, \nu_{11}^\star)$ in $(\mathcal{V}_s \times \mathcal{V}_u) \subseteq \mathcal{M}$ that share the same prefix $\nu_0^\star \in \mathcal{V}_s$ but have different suffixes $\nu_{01}^\star, \nu_{11}^\star \in \mathcal{V}_u$.

This tuple is computed in H_{0-1} by rewinding (up to t times) the execution of A_2. Specifically, consider a session identifier $i \in \{0, 1\}$, representing the i-th session, under the condition that no rewinding has been performed prior to this session (i.e., $\mathsf{rwd} = 0$, meaning the i-th session is the first requiring rewinding). In this case, H_{0-1} assigns the tuple $(\sigma_0^\star, \sigma_1^\star, \nu_{01}^\star, \nu_{11}^\star, \nu_0^\star)$ to rwdmsg by rewinding the execution A_2 during this session. No rewinding occurs in the $(1-i)$-th session.

In H_{0-1}, the behavior of the oracle User_2 is modified as follows. When A_2 calls User_2 with input $(i, \tilde{\sigma})$, the oracle first checks whether no rewinding has occurred in the $(1-i)$-th session by verifying $\mathsf{rwd} = 0$. If no rewinding has taken place, User_2 sets $\mathsf{rwd} = 1$ and executes A_2 up to t times, using the same fixed random coins τ as in the "main" execution of A_2[21].

During these rewound executions, the break from the "main" execution happens when A_2 interacts with User_1 for session i. Here, the value ν_{u_i} is replaced with a fresh value $\nu'_{u_i} \leftarrow\!\!\$\ \mathcal{V}_u \setminus \{\nu_{u_i}\}$. Then when A_2 interacts with User_2: (a) If A_2 attempts to call User_2 for the $(1-i)$-th session, arriving at the point that another rewinding of A_2 should be performed, the execution of A_2 is terminated[22]. (b) Otherwise, if A_2 call User_2 for the i-th session, H_{0-1} collects the signature input $\tilde{\sigma}$, terminates A_2, and checks whether $\tilde{\sigma}'_i$ is a valid signature for the message (ν_{s_i}, ν'_{u_i}) under $\mathsf{vk}'_{\mathsf{Sch}}$. If the signature is valid or if t attempts have been exhausted, the rewinding process stops. Otherwise, the game restarts another (rewound) execution of A_2.

Thus, in H_{0-1}, two possible outcomes arise:

[21] Fixing the random coins of A_2 also implicitly fixes any probabilistic choices within the oracles. As a result, the execution of A_2, including its interactions with the oracles and their responses, remains deterministic unless explicitly stated otherwise.

[22] This happens because A_2 may call User_2 for a session that depends on the modified message ν'_{u_i} received from User_1, that is different in every rewinding.

Game $\mathrm{BLD}^{\mathsf{A},b}_{\mathsf{PBSch}[\mathsf{P}]}(\lambda)$, $\mathrm{H}^{\mathsf{A},b}_{0-1}$, $\mathrm{H}^{\mathsf{A},b}_{1}$, $\mathrm{H}^{\mathsf{A},b}_{2}$, $\mathrm{H}^{\mathsf{A},b}_{3}$

$(q, \mathbb{G}, G, \mathsf{H}_q) \leftarrow \mathsf{Sch.Setup}(1^\lambda)$
$(\mathsf{ek}, \mathsf{dk}) \leftarrow \mathsf{PKE.KeyGen}(1^\lambda)$
$\mathsf{par} := ((q, \mathbb{G}, G), \mathsf{H}_q, \mathsf{ek})$
$b_0 := b; b_1 := (1-b); \tau \leftarrow\!\!\$\ \{0,1\}^*$
$(\varphi_0, \varphi_1, m_0, m_1, (X, X'), \mathsf{st}) \leftarrow \mathsf{A}_1(\mathsf{par})$
if $\exists i, j \in \{0,1\} : \mathsf{P}(\varphi_i, m_j) = 0$ **then**
  return 0
$(\mathsf{sess}_0, \mathsf{sess}_1) := (\mathtt{init}, \mathtt{init})$
$(\mathsf{st}_0, \mathsf{st}_1) := (\bot, \bot)$
$\mathsf{rwd} := 0; \mathsf{rwdmsg} := \bot$ (H_{0-1})
$\bar{m} \leftarrow\!\!\$\ \{0,1\}^{|\mathcal{M}|}$ (H_2) $\bar{\bar{m}} \leftarrow\!\!\$\ \{0,1\}^{|\mathcal{M}|}$ (H_3)
$b' \leftarrow \mathsf{A}_2^{\mathrm{User}_0, \mathrm{User}_1, \mathrm{User}_2, \mathrm{User}_3}(\mathsf{st}; \tau)$
return $(b = b')$

Oracle $\mathrm{User}_0(i)$

if $i \notin \{0,1\} \vee \mathsf{sess}_i \neq \mathtt{init}$ **then**
  return $\bot$
$\mathsf{sess}_i = \mathsf{open}; (\alpha_i, \beta_i) \leftarrow\!\!\$\ \mathbb{Z}_q^2; \rho_i \leftarrow\!\!\$\ \mathcal{R}_{\mathsf{PKE}}$
$M := (m_{b_i}, \alpha_i, \beta_i)$
$M = (\bar{m}, 0, 0)$ (H_3)
$C_i := \mathsf{PKE.Enc}(\mathsf{ek}, M; \rho_i)$
$\mathsf{st}_i = (\alpha_i, \beta_i, \rho_i, C_i)$
return C_i

Oracle $\mathrm{User}_3(i, s)$

if $i \notin \{0,1\} \vee \mathsf{sess}_i \neq \mathtt{await}_2$ **then**
  return $\bot$
$\mathsf{sess}_i = \mathtt{closed}$
$(R_i, R'_i, \alpha_i, c_i) :\subseteq \mathsf{st}_i$
if $sG = R_i + c_i X$ **then**
  $\sigma_{b_i} = (R'_i, (s + \alpha_i))$
else $\sigma_{b_i} = \bot$
if $\mathsf{sess}_0 = \mathsf{sess}_1 = \mathtt{closed}$ **then**
  if $(\sigma_0 = \bot \vee \sigma_1 = \bot)$ **then**
    return $(\bot, \bot)$
  return (σ_0, σ_1)
return $(i, \mathtt{closed})$

Oracle $\mathrm{User}_1(i, (R, \nu_s))$

if $i \notin \{0,1\} \vee \mathsf{sess}_i \neq \mathtt{open}$ **then return** $\bot$
$\mathsf{sess}_i = \mathtt{await}_1; (\alpha_i, \beta_i, \rho_i, C_i) := \mathsf{st}_i$
$(R_i, \nu_{s_i}) := (R, \nu_s); \nu_{u_i} \leftarrow\!\!\$\ \mathcal{V}_u$
$\mathsf{st}_i = (\alpha_i, \beta_i, \rho_i, C_i, R_i, \nu_{s_i}, \nu_{u_i})$
return ν_{u_i}

Oracle $\mathrm{User}_2(i, \tilde{\sigma})$

if $i \notin \{0,1\} \vee \mathsf{sess}_i \neq \mathtt{await}_1$ **then return**
$\mathsf{sess}_i = \mathtt{await}_2; \mathsf{vk}'_{\mathsf{Sch}} := ((q, \mathbb{G}, G, \mathsf{H}_q), X')$
$(\alpha_i, \beta_i, \rho_i, C_i, R_i, \nu_{s_i}, \nu_{u_i}) := \mathsf{st}_i; \tilde{\sigma}_i := \tilde{\sigma}$
if $\mathsf{Sch.Verify}(\mathsf{vk}'_{\mathsf{Sch}}, (\nu_{s_i}, \nu_{u_i}), \tilde{\sigma}_i) \neq 1$ **then**
  return $\bot$
$R'_i := R_i + \alpha_i G + \beta_i X$
$c_i := \mathsf{H}_q(R'_i, X, m_{b_i}) + \beta_i$
$\mathsf{st}_i = (\alpha_i, \beta_i, \rho_i, C_i, R_i, \nu_{s_i}, \nu_{u_i}, R'_i, c_i)$
$g_i \leftarrow\!\!\$\ \{0,1\}^{|\tilde{\sigma}_i|}; \nu_{g_i} \leftarrow\!\!\$\ \mathcal{V}_u; \varrho \leftarrow\!\!\$\ \mathcal{R}_{\mathsf{PKE}}$
$\mathsf{wtn} := (m_{b_i}, \alpha_i, \beta_i, \rho_i, \tilde{\sigma}_i, g_i, \nu_{u_i}, \nu_{g_i}, \nu_{s_i}, \varrho)$
$N := (\tilde{\sigma}_i, g_i, \nu_{u_i}, \nu_{g_i}, \nu_{s_i})$
if $\mathsf{rwd} = 0$ **then** // no rewind done yet
  set $\mathsf{rwd} = 1$ **and run** $\mathsf{A}_2(\mathsf{st}; \tau)$
  at most t **times**
  stop if $\mathsf{Sch.Verify}(\mathsf{vk}'_{\mathsf{Sch}}, (\nu_{s_i}, \nu'_{u_i}), \tilde{\sigma}'_i) \neq 0$
  in session i **or if** t **is reached**
    // at each run, output a different value ν'_{u_i}
    // from User_1 and collect the signature $\tilde{\sigma}'_i$
    // that A_2 sends to User_2
  if t **is not reached then**
    $\mathsf{rwdmsg} = (\tilde{\sigma}_i, \tilde{\sigma}'_i, \nu_{u_i}, \nu'_{u_i}, \nu_{s_i})$
  else halt the run with 0
else if $\mathsf{rwdmsg} = \bot$ **then stop**
$(\sigma^\star_0, \sigma^\star_1, \nu^\star_{01}, \nu^\star_{11}, \nu^\star_0) := \mathsf{rwdmsg}$ (H_{0-1})
$N = (\sigma^\star_0, \sigma^\star_1, \nu^\star_{01}, \nu^\star_{11}, \nu^\star_0)$ (H_1)
$S := \mathsf{PKE.Enc}(\mathsf{ek}, N; \varrho)$
$\mathsf{stm} := (X, X', R_i, c_i, C_i, \varphi_i, \mathsf{ek}, S)$
$\mathsf{wtn} = (\bar{m}, 0, 0, \varrho, \sigma^\star_0, \sigma^\star_1, \nu^\star_{01}, \nu^\star_{11}, \nu^\star_0, \varrho)$ (H_2)
$\pi \leftarrow \mathsf{Niwi.Prove}(\mathsf{stm}, \mathsf{wtn})$
return (c_i, π, S)

Fig. 22. The BLD game from Fig. 2 for the scheme $\mathsf{PBSch}[\mathsf{P}, \mathsf{GrGen}, \mathsf{HGen}, \mathsf{PKE}, \mathsf{Niwi}]$ from Fig. 3 and hybrid games used in the proof of Thm. 2. H_i includes all boxes with an index $\leq i$ and ignores all boxes with and index $> i$.

1. **Rewinding fails:** The rewinding process terminates because t attempts have been exhausted without obtaining a valid signature. Since rewinding is limited to t attempts, failure implies that A_2 does not output a valid signature across t different executions, where each execution samples a new value ν'_{u_i}. This could be due to either an invalid signature or A_2 requesting interaction with User_2 for session $(1-i)$, arriving at the point that another rewinding of A_2 should be performed. In this case, H_{0-1} returns 0.
2. **Rewinding succeeds:** The process halts after obtaining a valid signature. Let $\tilde{\sigma}_i$ be the signature obtained from the initial (non-rewound) execution of session i, which verifies the message $(\nu_{s_i}, \nu_{u_i}) \in \mathcal{M}$. Similarly, let $\tilde{\sigma}'_i$ be the valid signature produced during rewinding for (ν_{s_i}, ν'_{u_i}). The game then sets: $\mathsf{rwdmsg} = (\tilde{\sigma}_i, \tilde{\sigma}'_i, \nu_{u_i}, \nu'_{u_i}, \nu_{s_i})$.
   This assignment ensures that rwdmsg contains exactly two valid signatures for two distinct messages in $(\mathcal{V}_s \times \mathcal{V}_u) \subseteq \mathcal{M}$ that share the same prefix but differ in their suffixes.

We analyze the games H_0 and H_{0-1}. The only distinction between these games arises when the rewinding process fails; specifically, in H_{0-1}, after t attempts, the game is unable to compute a valid tuple to assign to rwdmsg. In all other scenarios, the games are identical. We now argue that for a polynomial number of rewinding attempts, and thus a polynomial-time execution of the game H_{0-1}, the rewinding process fails only with negligible probability. Our analysis follows the approach of Canetti et al. [CGGM00], but in a simpler setting, as our protocol does not involve prover resettability (i.e., the user in our case) or a polynomial number of signing keys. This analysis remains valid for the adversaries we construct in subsequent reductions, and for this reason, we do not repeat later.

Let t denote the number of rewind attempts of A. Consider the probability p of the event E:

> E occurs when, during the rewinding of A, the adversary receives a value from the set $\mathcal{V}_u \setminus \{\nu_u\}$ and subsequently requests to play the i-th session, where the rewinding begins.

In other words, E signifies that, upon being rewound and receiving a uniformly sampled value from $\mathcal{V}_u \setminus \{\nu_u\}$ via the oracle User_1, the adversary A interacts with User_2 for the i-th session. Before engaging in other sessions and reaching another rewinding point, A provides a valid signature to User_2 for a message with the same prefix as in the original execution, but with a different suffix sampled from $\mathcal{V}_u \setminus \{\nu_u\}$ during rewinding. To ensure that a second valid signature is obtained from the adversary in H_{0-1}, the number of rewinding attempts must be approximately λ/p. We analyze two cases:

- If p is non-negligible, say $p = 1/\mathsf{poly}(\lambda)$, then the game H_{0-1} runs in $\mathcal{O}(\lambda \cdot \mathsf{poly}(\lambda))$ time.
- If p is negligible, then obtaining a second signature would require super-polynomial time (since $1/p$ is super-polynomial). In this case, H_{0-1} terminates after a polynomial number of attempts, returning $\perp$.

This abort event in $\mathsf{PBSch.Sim}$ occurs only if E happens with negligible probability (i.e., p is negligible). That is, after λ/p uniform samples from $\mathcal{V}_u \setminus \{\nu_u\}$, the adversary A fails to request the i-th session with User_2 and produce a valid signature. This implies that A requests the i-th session "only" when it receives ν_u, meaning the probability of making such a request is $1/(\lambda/p)$, which is negligible since p is negligible. Thus, the probability that H_{0-1} aborts is negligible, implying that the two games H_0 and H_{0-1} differ only with negligible probability.

Game H_1. The game H_1 is identical to game H_{0-1}, except for the behavior of the oracle User_2. Specifically, in User_2, instead of computing the encryption S, to be returned to A_2, of the message $N = (\tilde{\sigma}_i, g_i, \nu_{u_i}, \nu_{g_i}, \nu_{s_i})$, we compute the encryption of a fixed message that is the one in rwdmsg. Namely, we compute the encryption with the message $(\sigma^\star_0, \sigma^\star_1, \nu^\star_{01}, \nu^\star_{11}, \nu^\star_0)$ in every session.

Reduction from CPA-security of PKE. We show that the difference between the advantage $\mathsf{Adv}^{\text{H}_{0-1}}(\mathsf{A}, \lambda)$ and the advantage $\mathsf{Adv}^{\text{H}_1}(\mathsf{A}, \lambda)$ is bounded by the advantage of winning the CPA game (Fig. 6) played by the adversaries M_0 and M_1 (in Fig. 23) against the PKE scheme.

According to the definition of the CPA game, M_b, with $b \in \{0, 1\}$, receives as input ek generated by $\mathsf{PKE.KeyGen}$. With this, M_b simulates the game $\text{H}^{\mathsf{A},b}_{0-1}$ to A, using its oracle Enc. Namely, during a call of User_2 from A, M_b sets $N := (\tilde{\sigma}_i, g_i, \nu_{u_i}, \nu_{g_i}, \nu_{s_i})$ and $N' := (\sigma^\star_0, \sigma^\star_1, \nu^\star_{01}, \nu^\star_{11}, \nu^\star_0)$, where N' is the same message for every session, while N is a session-dependent message, then M_b calls its encryption oracle on (N, N') and uses the received ciphertext in stm and outputs the received ciphertext as part of the output

Game $\mathsf{M}_b^{\mathrm{Enc}}(\mathsf{ek})$

1 : $1^\lambda :\subseteq \mathsf{ek}; (q, \mathbb{G}, G, \mathsf{H}_q) \leftarrow \mathsf{Sch.Setup}(1^\lambda)$
2 : $\mathsf{par} := ((q, \mathbb{G}, G), \mathsf{H}_q, \mathsf{ek})$
3 : $\tau \leftarrow\!\!\$\ \{0,1\}^*$
4 : $(\varphi_0, \varphi_1, m_0, m_1, (X, X'), \mathsf{st}) \leftarrow \mathsf{A}_1(\mathsf{par})$
5 : **if** $\exists i, j \in \{0,1\} : \mathsf{P}(\varphi_i, m_j) = 0$ **then**
6 : **return** 0
7 : $(\mathsf{sess}_0, \mathsf{sess}_1) := (\mathtt{init}, \mathtt{init})$
8 : $(\mathsf{st}_0, \mathsf{st}_1) := (\bot, \bot)$
$\mathsf{rwd} := 0; \mathsf{rwdmsg} := \bot$ (H_{0-1})
9 : $b' \leftarrow \mathsf{A}_2^{\mathrm{User}_0, \mathrm{User}_1, \mathrm{User}_2, \mathrm{User}_3}(\mathsf{st}; \tau)$
10 : **return** b'

Oracle $\mathrm{User}_2(i, \tilde{\sigma})$

1 : **if** $i \notin \{0,1\} \vee \mathsf{sess}_i \neq \mathtt{await}_1$ **then return**
2 : $\mathsf{sess}_i = \mathtt{await}_2; \mathsf{vk}'_{\mathsf{Sch}} := ((q, \mathbb{G}, G, \mathsf{H}_q), X')$
3 : $(\alpha_i, \beta_i, \rho_i, C_i, R_i, \nu_{s_i}, \nu_{u_i}) := \mathsf{st}_i; \tilde{\sigma}_i := \tilde{\sigma}$
4 : **if** $\mathsf{Sch.Verify}(\mathsf{vk}'_{\mathsf{Sch}}, (\nu_{s_i}, \nu_{u_i}), \tilde{\sigma}_i) \neq 1$ **then**
5 : **return** $\bot$
6 : $R'_i := R_i + \alpha_i G + \beta_i X$
7 : $c_i := \mathsf{H}_q(R'_i, X, m_{b_i}) + \beta_i$
8 : $\mathsf{st}_i = (\alpha_i, \beta_i, \rho_i, C_i, R_i, \nu_{s_i}, \nu_{u_i}, R'_i, c_i)$
9 : $g_i \leftarrow\!\!\$\ \{0,1\}^{|\tilde{\sigma}_i|}; \nu_{g_i} \leftarrow\!\!\$\ \mathcal{V}_u; \varrho \leftarrow\!\!\$\ \mathcal{R}_{\mathsf{PKE}}$
10 : $\mathsf{wtn} := (m_{b_i}, \alpha_i, \beta_i, \rho_i, \tilde{\sigma}_i, g_i, \nu_{u_i}, \nu_{g_i}, \nu_{s_i}, \varrho)$
11 : $N := (\tilde{\sigma}_i, g_i, \nu_{u_i}, \nu_{g_i}, \nu_{s_i})$
if $\mathsf{rwd} = 0$ **then** // no rewind done yet
set $\mathsf{rwd} = 1$ **and run** $\mathsf{A}_2(\mathsf{st}; \tau)$
at most t **times**
stop if $\mathsf{Sch.Verify}(\mathsf{vk}'_{\mathsf{Sch}}, (\nu_{s_i}, \nu'_{u_i}), \tilde{\sigma}'_i) \neq 0$
in session i **or if** t **is reached**
// at each run, output a different value ν'_{u_i}
// from User_1 and collect the signature $\tilde{\sigma}'_i$
// that A_2 sends to User_2
if t **is not reached then**
$\mathsf{rwdmsg} = (\tilde{\sigma}_i, \tilde{\sigma}'_i, \nu_{u_i}, \nu'_{u_i}, \nu_{s_i})$
else halt the run with 0
else if $\mathsf{rwdmsg} = \bot$ **then stop**
$(\sigma^\star_0, \sigma^\star_1, \nu^\star_{01}, \nu^\star_{11}, \nu^\star_0) := \mathsf{rwdmsg}$ (H_{0-1})
$N = (\sigma^\star_0, \sigma^\star_1, \nu^\star_{01}, \nu^\star_{11}, \nu^\star_0)$ (H_1)
12 : $S \leftarrow \mathrm{Enc}(N, N')$
13 : $\mathsf{stm} := (X, X', R_i, c_i, C_i, \varphi_i, \mathsf{ek}, S)$
14 : $\pi \leftarrow \mathsf{Niwi.Prove}(\mathsf{stm}, \mathsf{wtn})$
15 : **return** (c_i, π, S)

Fig. 23. M_b paying against CPA security of PKE. The oracles User_0, User_1 and User_3 are simulated to A as defined in game H_1 in Fig. 22. In User_2, specifically at line 10, the variable b_i is used as a shorthand. Depending on the value of i, b_i is defined as $b_0 = b$ or $b_1 = (1 - b)$, as also done in Fig. 22.

of oracle User_2. By construction of M_b we have that $\Pr\left[\text{CPA}^{\mathsf{M}_b,0}_{\mathsf{PKE}}(\lambda)=1\right] = \Pr\left[\text{H}^{\mathsf{A},b}_{0-1}(\lambda)=1\right]$ and also that $\Pr\left[\text{CPA}^{\mathsf{M}_b,1}_{\mathsf{PKE}}(\lambda)=1\right] = \Pr\left[\text{H}^{\mathsf{A},b}_{1}(\lambda)=1\right]$ and therefore:

$$\begin{aligned}\mathsf{Adv}^{\text{CPA}}_{\mathsf{PKE}}(\mathsf{M}_b,\lambda) &:= \left|\Pr\left[\text{CPA}^{\mathsf{M}_b,0}_{\mathsf{PKE}}(\lambda)=1\right] - \Pr\left[\text{CPA}^{\mathsf{M}_b,1}_{\mathsf{PKE}}(\lambda)=1\right]\right| \\ &= \left|\Pr\left[\text{H}^{\mathsf{A},b}_{0-1}(\lambda)=1\right] - \Pr\left[\text{H}^{\mathsf{A},b}_{1}(\lambda)=1\right]\right|\end{aligned}$$

Together with the triangular inequality, this yields:

$$\begin{aligned}\mathsf{Adv}^{\text{H}_{0-1}}(\mathsf{A},\lambda) &:= \left|\Pr\left[\text{H}_{0-1}{}^{\mathsf{A},1}(\lambda)=1\right] - \Pr\left[\text{H}_{0-1}{}^{\mathsf{A},0}(\lambda)=1\right]\right| \\ &= \left|\Pr\left[\text{H}_{0-1}{}^{\mathsf{A},1}(\lambda)=1\right] - \Pr\left[\text{H}_{0-1}{}^{\mathsf{A},0}(\lambda)=1\right] + \Pr\left[\text{H}^{\mathsf{A},0}_{1}(\lambda)=1\right]\right. \\ &\quad \left. - \Pr\left[\text{H}^{\mathsf{A},0}_{1}(\lambda)=1\right] + \Pr\left[\text{H}^{\mathsf{A},1}_{1}(\lambda)=1\right] - \Pr\left[\text{H}^{\mathsf{A},1}_{1}(\lambda)=1\right]\right| \\ &\leq \mathsf{Adv}^{\text{CPA}}_{\mathsf{PKE}}(\mathsf{M}_0,\lambda) + \mathsf{Adv}^{\text{CPA}}_{\mathsf{PKE}}(\mathsf{M}_1,\lambda) + \\ &\quad + \left|\Pr\left[\text{H}^{\mathsf{A},1}_{1}(\lambda)=1\right] - \Pr\left[\text{H}^{\mathsf{A},0}_{1}(\lambda)=1\right]\right|\end{aligned}$$

Namely, that:

$$\mathsf{Adv}^{\text{H}_{0-1}}(\mathsf{A},\lambda) \leq \mathsf{Adv}^{\text{CPA}}_{\mathsf{PKE}}(\mathsf{M}_0,\lambda) + \mathsf{Adv}^{\text{CPA}}_{\mathsf{PKE}}(\mathsf{M}_1,\lambda) + \mathsf{Adv}^{\text{H}_1}(\mathsf{A},\lambda) \tag{9}$$

Game H_2. The game H_2 is identical to game H_1, except for the sampling of a message $\bar{m} \leftarrow\!\$\ \{0,1\}^{|\mathcal{M}|}$ and for the behavior of the oracle User_2. Specifically, in User_2, instead of computing the proof π, to be returned to A_2, using the witness $\mathsf{wtn} = (m_{b_i}, \alpha_i, \beta_i, \rho_i, \tilde{\sigma}_i, g_i, \nu_{u_i}, \nu_{g_i}, \nu_{s_i}, \varrho)$, we compute the proof using an alternative witness $\mathsf{wtn}' = (\bar{m}, 0, 0, \varrho, \sigma^\star_0, \sigma^\star_1, \nu^\star_{01}, \nu^\star_{11}, \nu^\star_0, \varrho)$. By construction, this witness wtn' is valid for the statement $\mathsf{stm} = (X, X', R_i, c_i, C_i, \varphi_i, \mathsf{ek}, S)$ under the relation $\mathsf{R}_{\mathsf{PBSch}}$. This validity holds because the following conditions are satisfied:

$$\begin{gathered}S = \mathsf{PKE.Enc}(\mathsf{ek}, (\sigma^\star_0, \sigma^\star_1, \nu^\star_{01}, \nu^\star_{11}, \nu^\star_0); \varrho) \wedge \nu^\star_{01} \neq \nu^\star_{11} \wedge \\ 1 = \mathsf{Sch.Verify}(\mathsf{vk}'_{\mathsf{Sch}}, (\nu^\star_0, \nu^\star_{01}), \sigma^\star_0) \wedge 1 = \mathsf{Sch.Verify}(\mathsf{vk}'_{\mathsf{Sch}}, (\nu^\star_0, \nu^\star_{11}), \sigma^\star_1)\end{gathered}$$

where $\mathsf{vk}'_{\mathsf{Sch}} = ((q, \mathbb{G}, G, \mathsf{H}_q), X')$.

Reduction to the WI of Niwi. We show that the difference between the advantage $\mathsf{Adv}^{\text{H}_1}(\mathsf{A},\lambda)$ and the advantage $\mathsf{Adv}^{\text{H}_2}(\mathsf{A},\lambda)$ is bounded by the advantage of winning the WI game (Fig. 11) played by the adversaries W_0 and W_1, with access to the oracle Prove (Fig. 24), against the Niwi. According to the definition of the WI game, W_b, where $b \in \{0,1\}$, receives as input $\mathsf{par}_{\mathsf{R}}$ generated by Niwi.Rel. Based on PBSch in Fig. 3, $\mathsf{par}_{\mathsf{R}}$ is defined as $(q, \mathbb{G}, G, \mathsf{H}_q)$. Using this input, W_b simulates the game $\text{H}^{\mathsf{A},b}_1$ for A, leveraging its oracle Prove. The only difference introduced by W_b lies in how the proof π is computed. Specifically, W_b first computes an alternative valid witness wtn' for the statement stm. Then, it queries its oracle Prove to compute π using the statement stm and both witnesses, wtn and wtn'. By the construction of W_b, it holds that $\Pr\left[\text{WI}^{\mathsf{W}_b,0}_{\mathsf{Niwi}}(\lambda)=1\right] = \Pr\left[\text{H}^{\mathsf{A},b}_{1}(\lambda)=1\right]$ and also that $\Pr\left[\text{WI}^{\mathsf{W}_b,1}_{\mathsf{Niwi}}(\lambda)=1\right] = \Pr\left[\text{H}^{\mathsf{A},b}_{2}(\lambda)=1\right]$ and therefore:

$$\begin{aligned}\mathsf{Adv}^{\text{WI}}_{\mathsf{Niwi}}(\mathsf{W}_b,\lambda) &:= \left|\Pr\left[\text{WI}^{\mathsf{W}_b,0}_{\mathsf{Niwi}}(\lambda)=1\right] - \Pr\left[\text{WI}^{\mathsf{W}_b,1}_{\mathsf{Niwi}}(\lambda)=1\right]\right| \\ &= \left|\Pr\left[\text{H}^{\mathsf{A},b}_{1}(\lambda)=1\right] - \Pr\left[\text{H}^{\mathsf{A},b}_{2}(\lambda)=1\right]\right|\end{aligned}$$

Together with the triangular inequality, this yields:

$$\begin{aligned}\mathsf{Adv}^{\text{H}_1}(\mathsf{A},\lambda) &:= \left|\Pr\left[\text{H}_1{}^{\mathsf{A},1}(\lambda)=1\right] - \Pr\left[\text{H}_1{}^{\mathsf{A},0}(\lambda)=1\right]\right| \\ &= \left|\Pr\left[\text{H}_1{}^{\mathsf{A},1}(\lambda)=1\right] - \Pr\left[\text{H}_1{}^{\mathsf{A},0}(\lambda)=1\right] + \Pr\left[\text{H}^{\mathsf{A},0}_{2}(\lambda)=1\right] + \right. \\ &\quad \left. - \Pr\left[\text{H}^{\mathsf{A},0}_{2}(\lambda)=1\right] + \Pr\left[\text{H}^{\mathsf{A},1}_{2}(\lambda)=1\right] - \Pr\left[\text{H}^{\mathsf{A},1}_{2}(\lambda)=1\right]\right| \\ &\leq \mathsf{Adv}^{\text{WI}}_{\mathsf{Niwi}}(\mathsf{W}_0,\lambda) + \mathsf{Adv}^{\text{WI}}_{\mathsf{Niwi}}(\mathsf{W}_1,\lambda) + \\ &\quad + \left|\Pr\left[\text{H}^{\mathsf{A},1}_{2}(\lambda)=1\right] - \Pr\left[\text{H}^{\mathsf{A},0}_{2}(\lambda)=1\right]\right|\end{aligned}$$

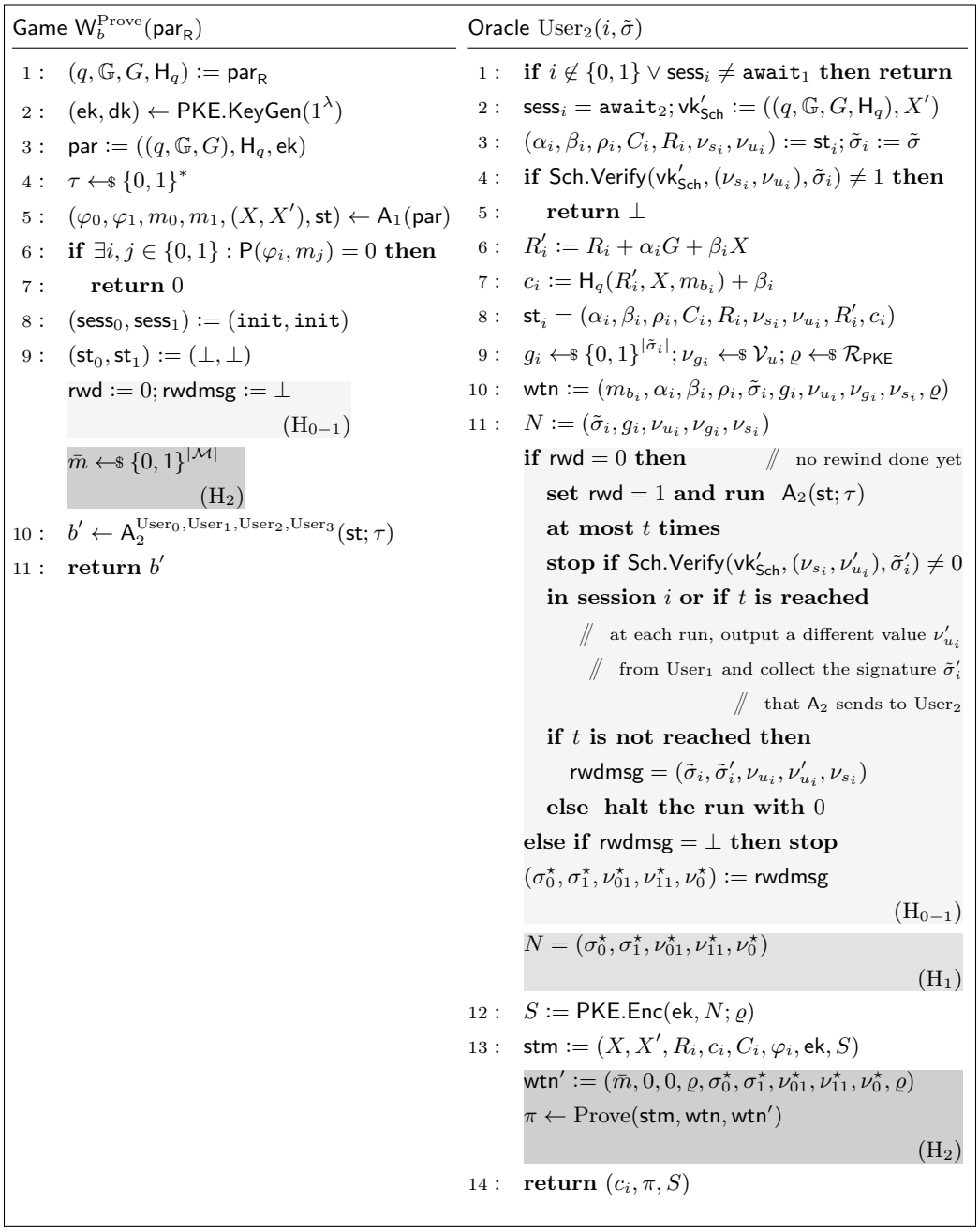


Fig. 24. W_b paying against WI security of $\mathsf{Niwi}[\mathsf{R}_{\mathsf{PBSch}}]$. The oracles User_0, User_1 and User_3 are simulated to A as defined in game H_2 in Fig. 22. In User_2 the variable b_i is used as a shorthand. Depending on the value of i, b_i is defined as $b_0 = b$ or $b_1 = (1 - b)$, as also done in Fig. 22.

Game $\mathsf{T}_b^{\mathrm{Enc}}(\mathsf{ek})$	Oracle $\mathrm{User}_0(i)$
$1^\lambda :\subseteq \mathsf{ek}; (q, \mathbb{G}, G, \mathsf{H}_q) := \mathsf{Sch.Setup}(1^\lambda)$	**if** $i \notin \{0,1\} \vee \mathsf{sess}_i \neq \mathtt{init}$
$\mathsf{par} := ((q, \mathbb{G}, G), \mathsf{H}_q, \mathsf{ek})$	**then return** $\perp$
$\tau \leftarrow\!\!\$\ \{0,1\}^*$	$\mathsf{sess}_i = \mathtt{open}$
$(\varphi_0, \varphi_1, m_0, m_1, (X, X'), \mathsf{st}) \leftarrow \mathsf{A}_1(\mathsf{par})$	$(\alpha_i, \beta_i) \leftarrow\!\!\$\ \mathbb{Z}_q^2$
if $\exists i, j \in \{0,1\} : \mathsf{P}(\varphi_i, m_j) = 0$ **then**	$\rho_i \leftarrow\!\!\$\ \mathcal{R}_{\mathsf{PKE}}$
return 0	$M := (m_{b_i}, \alpha_i, \beta_i)$
$(\mathsf{sess}_0, \mathsf{sess}_1) := (\mathtt{init}, \mathtt{init})$	$M' := (\bar{\bar{m}}, 0, 0)$ (H_3)
$(\mathsf{st}_0, \mathsf{st}_1) := (\perp, \perp)$	$C_i \leftarrow \mathrm{Enc}(M, M')$
$\mathsf{rwd} := \perp; \mathsf{rwdmsg} := \perp$ (H_{0-1})	$\mathsf{st}_i = (\alpha_i, \beta_i, \rho_i, C_i)$
$\bar{m} \leftarrow\!\!\$\ \{0,1\}^{\lvert\mathcal{M}\rvert}$ (H_2) $\quad \bar{\bar{m}} \leftarrow\!\!\$\ \{0,1\}^{\lvert\mathcal{M}\rvert}$ (H_3)	**return** C_i
$b' \leftarrow \mathsf{A}_2^{\mathrm{User}_0, \mathrm{User}_1, \mathrm{User}_2, \mathrm{User}_3}(\mathsf{st}; \tau)$	
return b'	

Fig. 25. T_b paying against CPA security of PKE. The oracles User_1, User_2 and User_3 are simulated to A as defined in game H_3 in Fig. 22.

Namely, that:

$$\mathsf{Adv}^{\mathrm{H}_1}(\mathsf{A}, \lambda) \leq \mathsf{Adv}^{\mathrm{WI}}_{\mathsf{Niwi}}(\mathsf{W}_0, \lambda) + \mathsf{Adv}^{\mathrm{WI}}_{\mathsf{Niwi}}(\mathsf{W}_1, \lambda) + \mathsf{Adv}^{\mathrm{H}_2}(\mathsf{A}, \lambda) \tag{10}$$

Game H_3**.** The game H_3 is identical to game H_2, except for sampling a message $\bar{\bar{m}} \leftarrow\!\!\$\ \{0,1\}^{|\mathcal{M}|}$ and the behavior of the oracle User_0. Specifically, in User_0, the message M, namely the plaintext that is then encrypted in C_i and returned to A_2, is now a fixed message that is always the same for every played session and is defined as $M := (\bar{\bar{m}}, 0, 0)$.

Reduction from CPA-security of PKE**.** We show that the difference between the advantage $\mathsf{Adv}^{\mathrm{H}_2}(\mathsf{A}, \lambda)$ and the advantage $\mathsf{Adv}^{\mathrm{H}_3}(\mathsf{A}, \lambda)$ is bounded by the advantage of winning the CPA game (Fig. 6) played by the adversaries T_0 and T_1 (in Fig. 25) against the PKE scheme. By construction of T_b we have that $\Pr\left[\mathrm{CPA}^{\mathsf{T}_b,0}_{\mathsf{PKE}}(\lambda) = 1\right] = \Pr\left[\mathrm{H}_2^{\mathsf{A},b}(\lambda) = 1\right]$ and also that $\Pr\left[\mathrm{CPA}^{\mathsf{T}_b,1}_{\mathsf{PKE}}(\lambda) = 1\right] = \Pr\left[\mathrm{H}_3^{\mathsf{A},b}(\lambda) = 1\right]$ and therefore:

$$\begin{aligned}\mathsf{Adv}^{\mathrm{CPA}}_{\mathsf{PKE}}(\mathsf{T}_b, \lambda) &:= \left|\Pr\left[\mathrm{CPA}^{\mathsf{T}_b,0}_{\mathsf{PKE}}(\lambda) = 1\right] - \Pr\left[\mathrm{CPA}^{\mathsf{T}_b,1}_{\mathsf{PKE}}(\lambda) = 1\right]\right| \\ &= \left|\Pr\left[\mathrm{H}_2^{\mathsf{A},b}(\lambda) = 1\right] - \Pr\left[\mathrm{H}_3^{\mathsf{A},b}(\lambda) = 1\right]\right|\end{aligned}$$

Together with the triangular inequality, this yields:

$$\begin{aligned}\mathsf{Adv}^{\mathrm{H}_2}(\mathsf{A}, \lambda) &:= \left|\Pr\left[\mathrm{H}_2{}^{\mathsf{A},1}(\lambda) = 1\right] - \Pr\left[\mathrm{H}_2{}^{\mathsf{A},0}(\lambda) = 1\right]\right| \\ &= \left|\Pr\left[\mathrm{H}_2{}^{\mathsf{A},1}(\lambda) = 1\right] - \Pr\left[\mathrm{H}_2{}^{\mathsf{A},0}(\lambda) = 1\right] + \Pr\left[\mathrm{H}_3^{\mathsf{A},0}(\lambda) = 1\right] + \right. \\ &\qquad \left. - \Pr\left[\mathrm{H}_3^{\mathsf{A},0}(\lambda) = 1\right] + \Pr\left[\mathrm{H}_3^{\mathsf{A},1}(\lambda) = 1\right] - \Pr\left[\mathrm{H}_3^{\mathsf{A},1}(\lambda) = 1\right]\right| \\ &\leq \mathsf{Adv}^{\mathrm{CPA}}_{\mathsf{PKE}}(\mathsf{T}_0, \lambda) + \mathsf{Adv}^{\mathrm{CPA}}_{\mathsf{PKE}}(\mathsf{T}_1, \lambda) + \\ &\quad + \left|\Pr\left[\mathrm{H}_3^{\mathsf{A},1}(\lambda) = 1\right] - \Pr\left[\mathrm{H}_3^{\mathsf{A},0}(\lambda) = 1\right]\right|\end{aligned}$$

Namely, that:

$$\mathsf{Adv}^{\mathrm{H}_2}(\mathsf{A}, \lambda) \leq \mathsf{Adv}^{\mathrm{CPA}}_{\mathsf{PKE}}(\mathsf{T}_0, \lambda) + \mathsf{Adv}^{\mathrm{CPA}}_{\mathsf{PKE}}(\mathsf{T}_1, \lambda) + \mathsf{Adv}^{\mathrm{H}_3}(\mathsf{A}, \lambda) \tag{11}$$

The signer's view after the successful completion of the two signing sessions consists of the parameters $\mathsf{par} := ((q, \mathbb{G}, G), \mathsf{H}_q, \mathsf{ek})$ and the signatures with the corresponding messages $(m_0, (R'_0, s'_0))$ and $(m_1, (R'_1, s'_1))$, as well as $\{(C_i, R_i, \nu_{s_i}, \nu_{u_i}, \tilde{\sigma}_i, c_i, \pi_i, S_i, s_i)_{i \in \{0,1\}}\}$ where $i = 0$ denotes values obtained in the first session, and for $i = 1$ values of the second session respectively. Since the ciphertext C_i is an encryption of fixed values, ν_{u_i} is uniformly sampled from $\mathcal{V}_u$, the ciphertext S_i is an encryption of fixed values and the proof π_i is computed on a witness with a fixed first message, they hold no information

on bit b. Consider the tuple $(m_j, (R'_j, s'_j))$ for $j \in \{0,1\}$, assume that it corresponds to session i with the tuple (R_i, c_i, s_i). By defining $\alpha := s'_j - s_i$, there exists a value β such that $R'_j = R_i + \alpha G + \beta X$. This implies that both session tuples, (R_0, c_0, s_0) and (R_1, c_1, s_1), provide valid explanations for (R'_j, s'_j). Hence, the advantage:

$$\mathsf{Adv}^{\mathrm{H}_3}(\mathsf{A}, \lambda) := \left| \Pr\left[\mathrm{H}_3^{\mathsf{A},1}(\lambda) = 1\right] - \Pr\left[\mathrm{H}_3^{\mathsf{A},0}(\lambda) = 1\right] \right| = 0 \tag{12}$$

Namely, the advantage in distinguish H_3 with $b = 0$, from H_3 with $b = 1$ is 0.

The proof now follows from Equations (9,10,11,12). □

D Proof of Theorem 4

We give a formal proof that our predicate blind signature scheme PBSch from Fig. 3 satisfies deniability as defined in Def. 4. In this section, we employ the keyword "**halt the run with**" inside an oracle to explicitly denote the termination of the game and its output, rather than the return value of the oracle.

For clarity, we begin by outlining the structure of the proof. To simplify the exposition, we first present the theorem in the case where the commitment Cmt is instantiated via a PKE scheme PKE, where the setup safely generates a key pair. This case is particularly convenient, as it lets us focus on the computation of the NIWI argument, which is the core part of our construction. We remark that when using another straight-line extractable commitment (e.g., the NPRO-based instantiation discussed above), there is no difference: such tools can be employed in a black-box manner.

Namely, as noted in App. B, we use the PKE scheme merely for convenience and to enhance readability; it can be smoothly substituted without any loss of details with the hiding property of a straight-line extractable commitment scheme (i.e., using such a tool in a black-box way instead of explicitly employing the PKE scheme).

Games H_R and H_S. The game H_R is the rDNB game from Fig. 4, where PBS is instantiated with PBSch as defined in Fig. 3. Specifically, the algorithms PBS.Setup, PBS.KeyGen, $\mathsf{PBS.User}_0$, $\mathsf{PBS.User}_1$ and $\mathsf{PBS.User}_2$ are replaced with their respective instantiations from Fig. 3. We describe the game H_R in Fig. 26.

According to the definition of deniability (Def. 4), a PBS scheme satisfying this property must include an additional algorithm PBS.Sim. The game H_S is the sDNB game from Fig. 4, where PBS is instantiated with PBSch as defined in Fig. 3 and thus there is the additional algorithm PBSch.Sim. We describe now how the algorithm PBSch.Sim works.

PBSch.Sim description. PBSch.Sim, on input $\mathsf{prds} := (\varphi_1, \ldots, \varphi_{\mathsf{poly}(\lambda)})$ and $\mathsf{par} := ((q, \mathbb{G}, G, \mathsf{H}_q), \mathsf{ek})$ generated according to the algorithm PBSch.Setup, simulates an execution between the adversary A (as defined in Def. 4, representing a malicious signer for a PBS scheme, in this case for PBSch) and a user, impersonated by PBSch.Sim. During the execution, PBSch.Sim first initializes three variables: $\mathsf{rwd} := 0$, $\mathsf{rwdmsg} := \bot$, and $key := \bot$, along with an empty list $\mathbf{S} := [\,]$. It then samples random coins $\tau \leftarrow\!\$\ \{0,1\}^*$ and executes A with inputs par, prds, and τ, i.e., $\mathsf{A}(\mathsf{par}, \mathsf{prds}; \tau)$.

PBSch.Sim returns exactly the same output that it receives from A. It also simulates calls to the oracles that A may request, as follows: (a) When A queries the Init oracle with the tuple (X, X'), PBSch.Sim performs the same steps as in Fig. 26. It first checks if the variable key is set to $\bot$. If not, it returns $\bot$. Otherwise, it assigns $key := (X, X')$ and returns this tuple to A. (b) When A queries User_0 with the session identifier i, PBSch.Sim checks if $\mathbf{S}_i$ is non-empty or if $key = \bot$. If either condition holds, it returns $\bot$ to A. Otherwise, PBSch.Sim samples a random binary string $\bar{m}$ with a length matching that of a message from the space $\mathcal{M}$ (known from the description of $\mathcal{M}$). It then computes $M := (\bar{m}, 0, 0)$ and encrypts this message using ek from par, namely, $C := \mathsf{PKE.Enc}(\mathsf{ek}, M)$. Finally, it updates $\mathbf{S}_i = (\mathtt{open}, C)$ and returns the ciphertext C to A. (c) When A queries User_1 with the session identifier i and a tuple (R, ν_s), PBSch.Sim checks if $\mathbf{S}_i[0] \neq \mathtt{open}$. If this is the case, it returns $\bot$ to A. Otherwise, PBSch.Sim uniformly samples a value $\nu_u \leftarrow\!\$\ \mathcal{V}_u$, updates $\mathbf{S}_i = (\mathtt{await}, C, \nu_s, \nu_u)$, and returns ν_u to A. (d) When A queries User_2 with the session identifier i and the value $\tilde{\sigma}$, PBSch.Sim checks if $\mathbf{S}_i[0] \neq \mathtt{await}$. If this condition is true, it returns $\bot$ to A. Otherwise, PBSch.Sim verifies whether $\tilde{\sigma}$ is a valid signature for the message (ν_s, ν_u), where such message is extracted from the values stored in $\mathbf{S}_i$, namely, $(\nu_s, \nu_u) :\subseteq \mathbf{S}_i$. The verification key is computed as $\mathsf{vk}'_{\mathsf{Sch}} := ((q, \mathbb{G}, G, \mathsf{H}_q), X')$, derived from $key = (X, X')$ and par. Specifically, PBSch.Sim checks if $\mathsf{Sch.Verify}(\mathsf{vk}'_{\mathsf{Sch}}, (\nu_s, \nu_u), \tilde{\sigma}) \neq 1$. If the verification fails, it returns $\bot$ to A. Otherwise, PBSch.Sim proceeds to the rewinding stage.

Rewinding Stage. PBSch.Sim first checks if the boolean value rwd is equal to 0, indicating that this is the first time during its interaction with A that it has reached the rewinding stage (i.e., the i-th session is the first to arrive at this stage).

If $\mathsf{rwd} = 0$, then PBSch.Sim runs A again, passing the same input and random coins as before (i.e., $\mathsf{A}(\mathsf{par}, \mathsf{prds}; \tau)$). PBSch.Sim interacts with A in exactly the same manner as in the initial execution for all oracle queries, returning the same outputs stored in $\mathbf{S}$ for each query (i.e., it does not compute a new output for the same input, it just returns the previous computed output). The only difference is how PBSch.Sim handles interactions with the oracle User_1 when A provides input $(i, (R, \nu_s))$, where i is the session identifier of the session that is in the rewinding stage.

During this oracle simulation, PBSch.Sim samples a different value $\nu'_u \leftarrow\!\!\$\ \mathcal{V}_u \setminus \{\nu_u\}$, where ν_u is the value output when A first queried User_1 for session i (i.e., $\nu_u :\subseteq \mathbf{S}_i$). PBSch.Sim returns ν'_u and continues responding to oracle queries made by A during this rewinding.

- If A, after observing the new value ν'_u, queries User_2 with a tuple where the first value $i' \neq i$, and PBSch.Sim needs to perform another rewinding stage for this new session while a rewinding is already ongoing, PBSch.Sim interrupts the execution of A.
- Otherwise, if A queries User_2 with a tuple where the first value $i' = i$, PBSch.Sim collects the input value $\tilde{\sigma}$ (that we rename $\tilde{\sigma}'$ for clarity).
  Then PBSch.Sim verifies if $\tilde{\sigma}'$ is a valid Schnorr signature under the verification key $\mathsf{vk}'_{\mathsf{Sch}}$, for the message (ν_s, ν'_u), where $\nu_s :\subseteq \mathbf{S}_i$. If the signature $\tilde{\sigma}'$ is valid, PBSch.Sim ends the rewinding stage and sets $\mathsf{rwdmsg} = (\tilde{\sigma}, \tilde{\sigma}', \nu_u, \nu'_u, \nu_s)$, where $(\tilde{\sigma}, \nu_u, \nu_s) :\subseteq \mathbf{S}_i$, and $(\tilde{\sigma}',\ \nu'_u)$ are the values collected during the rewinding stage (i.e., the accepting signature and the suffix of the message verified by such signature), otherwise if the signature $\tilde{\sigma}'$ is not a valid signaturePBSch.Sim interrupts the execution of A.

If the signature $\tilde{\sigma}'$ is not a valid signature, or if PBSch.Sim aborts the execution before collecting $\tilde{\sigma}'$, PBSch.Sim runs again A with the same input and the same fixed random coins (sampling a new value from $\mathcal{V}_u \setminus \{\nu_u\}$), PBSch.Sim makes up to t attempts.

Namely, the simulator PBSch.Sim repeats the same steps described above (i.e., the rewinding of A), except that it samples a new value ν'_u in each iteration. PBSch.Sim continues this process for at most t iterations, after which it terminates and aborts the simulation if unsuccessful. Specifically, PBS.Sim maintains a counter that increments with each new run of A (i.e., each "rewinding attempt"). In every run, PBS.Sim uniformly samples a value from $\mathcal{V}_u \setminus \{\nu_u\}$ and provides it to A (instead of $\nu_u :\subseteq \mathbf{S}_i$), aiming to obtain a new valid signature. This procedure is repeated for at most t attempts.

Then in case the **Rewinding Stage** ends without PBSch.Sim aborts, namely with $\mathsf{rwdmsg} \neq \bot$; PBSch.Sim, using the tuple from rwdmsg, computes the message $N := (\tilde{\sigma}, \tilde{\sigma}', \nu_u, \nu'_u, \nu_s)$ (note that N is always the same for all the sessions being simulated by PBSch.Sim for A). Next, it samples a value ϱ uniformly from $\mathcal{R}_{\mathsf{PKE}}$ and computes $S := \mathsf{PKE.Enc}(\mathsf{ek}, N; \varrho)$. Afterward, PBSch.Sim samples c uniformly from $\mathbb{Z}_q$ and computes $\mathsf{stm} := (X, X', R, c, C, \varphi_i, \mathsf{ek}, S)$, where $(R, C) :\subseteq \mathbf{S}_i$ (i.e., these are the values stored in $\mathbf{S}_i$ during the execution of User_0 and User_1 for the same session i). Next, PBSch.Sim computes the witness $\mathsf{wtn} := (\bar{m}, 0, 0, \varrho, \tilde{\sigma}, \tilde{\sigma}', \nu_u, \nu'_u, \nu_s, \varrho)$ and generates the proof $\pi \leftarrow \mathsf{Niwi.Prove}(\mathsf{stm}, \mathsf{wtn})$. Finally, it sets $\mathbf{S}_i = \texttt{closed}$ and returns the tuple (c, π, S) to A.

Simulator running time and abort condition. We analyze the running time of the simulator PBSch.Sim and the probability that it aborts during simulation. The only case where PBSch.Sim aborts is when it fails to obtain a second valid signature by rewinding the adversary A[23].We sketch the analysis here, as it is very similar to the one conducted by Canetti et al. in [CGGM00]. In fact, our analysis follows the same approach, but in a simpler setting, as our protocol does not involve resettability of the prover (the user in our case) or a polynomial number of signing keys.

First, observe that in PBSch.Sim, the number of rewind attempts is bounded by some value t. Adding this to the running time of PBSch.Sim for other computations, the total running time of the simulator is at most $\mathcal{O}(t \cdot \mathsf{poly}(\lambda))$, where the polynomial factor originates from the fact that, aside from rewinding, the simulator performs the same operations as an honest signer, which runs in $\mathcal{O}(\mathsf{poly}(\lambda))$ time by definition. The key point is to determine an appropriate bound on t. To do so, consider the probability p of the following event E:

> E occurs when, during the rewinding of A, the adversary receives a value from the set $\mathcal{V}_u \setminus \{\nu_u\}$ and subsequently requests to play the i-th session, where the rewinding begins.

[23] That is, we evaluate the case that PBSch.Sim cannot extract another valid signature through rewinding.

In other words, the event E means that when the adversary A is rewound and obtains a value uniformly sampled from $\mathcal{V}_u \setminus \{\nu_u\}$, from the oracle of the oracle User_1, A interact with the oracle User_2 for the i-th session (before interacting with User_2 in other sessions and arriving at a point where a new rewinding should be done), and A provides a valid signature as input to User_2 for a message with the same prefix as in the original execution of A, but a different suffix, which was sampled during the rewinding from $\mathcal{V}_u \setminus \{\nu_u\}$. To ensure that the simulator $\mathsf{PBSch.Sim}$ extracts a second valid signature, the simulator must run for $\mathcal{O}((\lambda/p) \cdot \mathsf{poly}(\lambda))$ time.

- If p is non-negligible, say $p = 1/\mathsf{poly}(\lambda)$, then the (expected) running time of $\mathsf{PBSch.Sim}$ is polynomial in the security parameter.
- If p is negligible, then $\mathsf{PBSch.Sim}$ should run in super-polynomial time (since $1/p$ is super-polynomial), meaning that the simulator can extract the second signature only after a super-polynomial time, and thus it aborts returning $\perp$.

This abort event of $\mathsf{PBSch.Sim}$ occurs only when E happens with negligible probability (i.e., the probability p that E happens is negligible). That is, for λ/p different uniform samples from $\mathcal{V}_u \setminus \{\nu_u\}$, the adversary A consistently fails to request the i-th session with User_2 and produce a valid signature. This implies that A requests the i-th session "only" when it receives ν_u, specifically that the probability of making such a request is $1/(\lambda/p)$, which is negligible since p is negligible. Thus, the probability that $\mathsf{PBSch.Sim}$ aborts is negligible.

Game H_{0-1}. In game H_{0-1}, we introduce two variables, rwd and rwdmsg, initialized respectively to 0 and $\perp$. Additionally, we modify the behavior of the User_2 oracle. Specifically, during an oracle call to User_2, the H_{0-1} assigns the variable rwdmsg the tuple $(\sigma_0^\star, \sigma_1^\star, \nu_{01}^\star, \nu_{11}^\star, \nu_0^\star)$ which is defined as follows. Given $\mathsf{vk}'_{\mathsf{Sch}} := ((q, \mathbb{G}, G, \mathsf{H}_q), X')$, we have $\mathsf{Sch.Verify}(\mathsf{vk}'_{\mathsf{Sch}}, (\nu_0^\star, \nu_{01}^\star), \sigma_0^\star) = 1$ and $\mathsf{Sch.Verify}(\mathsf{vk}'_{\mathsf{Sch}}, (\nu_0^\star, \nu_{11}^\star), \sigma_1^\star) = 1$. Concretely, N contains two signatures, $(\sigma_0^\star, \sigma_1^\star)$ and two messages in $(\mathcal{V}_s \times \mathcal{V}_u) \subseteq \mathcal{M}$ that share the same prefix $\nu_0^\star \in \mathcal{V}_s$ but have different suffixes, $\nu_{01}^\star, \nu_{11}^\star \in \mathcal{V}_u$.

Such a tuple is computed in H_{0-1} by rewinding A. Specifically, consider a session identifier i, representing the i-th session, under the condition that no rewinding has been performed prior to this session in H_{0-1} (i.e., $\mathsf{rwd} = 0$ and thus the i-th session is the first session that needs to rewind the execution of A). The tuple $(\sigma_0^\star, \sigma_1^\star, \nu_{01}^\star, \nu_{11}^\star, \nu_0^\star)$ is extracted during this i-th session by rewinding the execution of A. More formally, to rewind the execution of A, we modify User_2 in H_{0-1} to perform the exact same steps that were performed: (1) in H_{0-1} in the proof of blindness in App. C, as illustrated in Fig. 22; (2) in the **Rewinding Stage** of $\mathsf{PBSch.Sim}$. We will not formally repeat the process for computing the tuple $(\sigma_0^\star, \sigma_1^\star, \nu_{01}^\star, \nu_{11}^\star, \nu_0^\star)$ verbatim here, relying instead on the reader's understanding. Moreover, the game H_{0-1} is formally and clearly described in Fig. 26. The analysis demonstrating that H_R and H_{0-1} are statistically indistinguishable follows from that in App. C.

Game H_1. The game H_1 is identical to game H_{0-1}, except for the behavior of the oracle User_2. Specifically, in User_2, instead of computing the encryption S, to be returned to A_2, of the message $N = (\tilde{\sigma}_i, g_i, \nu_{u_i}, \nu_{g_i}, \nu_{s_i})$, we compute the encryption of a fixed message that is the one in rwdmsg. Namely, we compute the encryption with the message $(\sigma_0^\star, \sigma_1^\star, \nu_{01}^\star, \nu_{11}^\star, \nu_0^\star)$ in every session.

Reduction from CPA-security of PKE. We show that the advantage of a distinguisher D in distinguishing between the game H_{0-1} and the game H_1 is bounded by the advantage of winning the CPA game (Fig. 6), which is played by the adversary Q (Fig. 27) against the PKE scheme.

First, note that if the adversary A outputs a view that is (computationally) different in H_{0-1} and H_1, then there exists a distinguisher D that succeeds in distinguishing the two games. This must occur for some specific pair of tuples $\mathsf{msgs}^* := (m_1, \ldots, m_{\mathsf{poly}(\lambda)})$ and $\mathsf{prds}^* := (\varphi_1, \ldots, \varphi_{\mathsf{poly}(\lambda)})$. Conversely, if no such pair of tuples exists, then the two games are indistinguishable. Therefore, assuming that D succeeds in distinguishing the games, such a pair of tuples must necessarily exist. For this reason, we hardcode both msgs^* and prds^* into the algorithm of Q, allowing it to reuse these values when simulating H_{0-1} for A.

According to the definition of the CPA, Q receives as input ek generated by $\mathsf{PKE.KeyGen}$. With this, Q simulates the game H_{0-1} to A, using its oracle Enc. Namely, during a call of User_2 from A, Q sets $N := (\tilde{\sigma}_i, g_i, \nu_{u_i}, \nu_{g_i}, \nu_{s_i})$ and $N' := (\sigma_0^\star, \sigma_1^\star, \nu_{01}^\star, \nu_{11}^\star, \nu_0^\star)$, then Q calls its encryption oracle on (N, N') and uses the received ciphertext in stm and outputs the received ciphertext as part of the output of User_2. Finally, it passes the pair composed of par and the output of A to the distinguisher algorithm D and outputs the same output that the distinguisher outputs. By construction of Q we have that

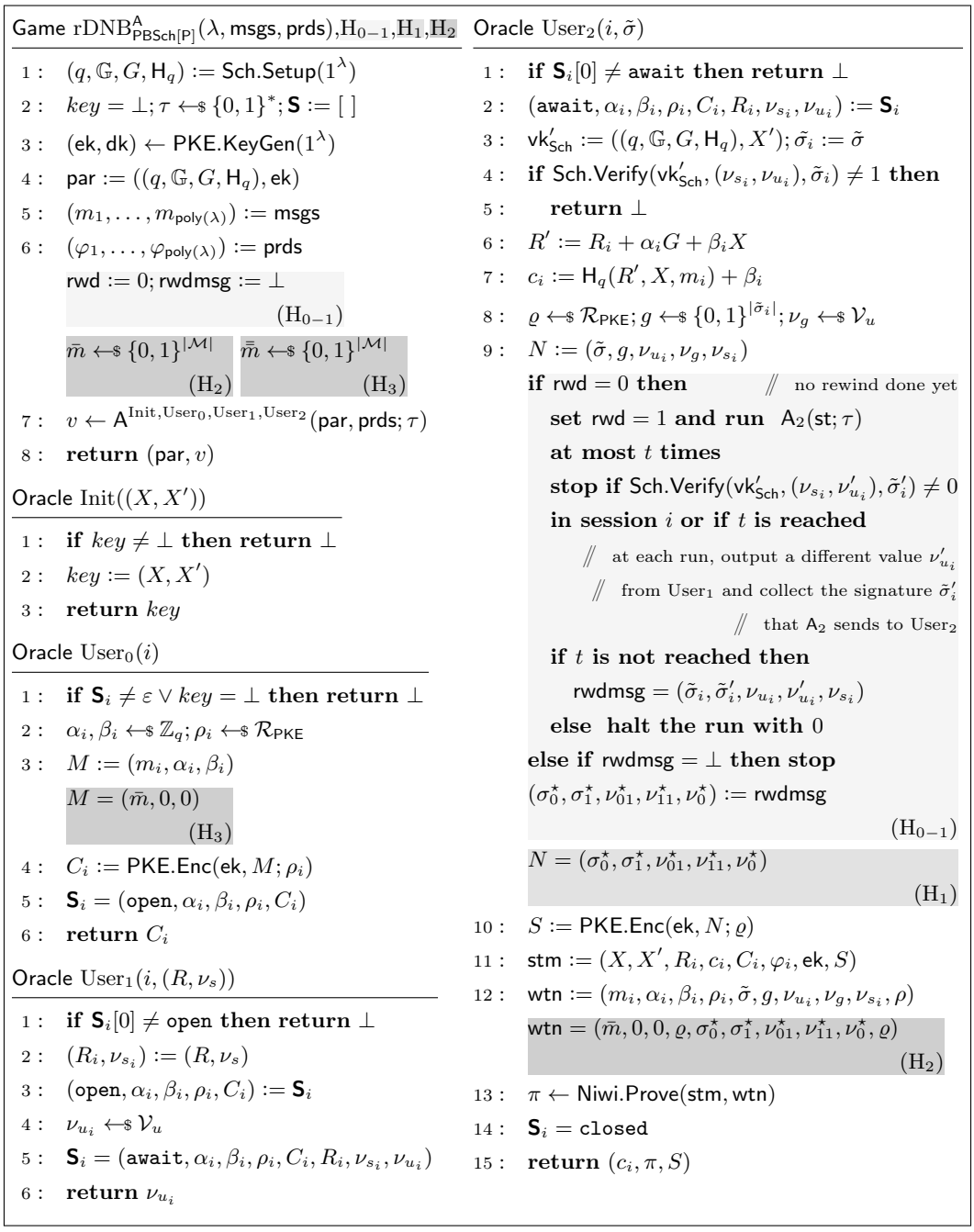


Fig. 26. The rDNB game from Fig. 4 for the scheme $\mathsf{PBSch}[\mathsf{P}, \mathsf{GrGen}, \mathsf{HGen}, \mathsf{PKE}, \mathsf{Niwi}]$ from Fig. 3 and hybrid games used in the proof of Thm. 2. H_i includes all boxes with an index $\leq i$ and ignores all boxes with and index $> i$.

$\Pr\left[\mathrm{CPA}^{\mathsf{Q},0}_{\mathsf{PKE}}(\lambda)=1\right] = \Pr\left[\mathsf{D}\left(\mathrm{H}^{\mathsf{A}}_{0-1}(\lambda)\right)=1\right]$ and also that $\Pr\left[\mathrm{CPA}^{\mathsf{Q},1}_{\mathsf{PKE}}(\lambda)=1\right] = \Pr\left[\mathsf{D}\left(\mathrm{H}^{\mathsf{A}}_{1}(\lambda)\right)=1\right]$ and therefore:

$$\begin{aligned}\mathsf{Adv}^{\mathrm{CPA}}_{\mathsf{PKE}}(\mathsf{Q},\lambda) &:= \left|\Pr\left[\mathrm{CPA}^{\mathsf{Q},0}_{\mathsf{PKE}}(\lambda)=1\right]-\Pr\left[\mathrm{CPA}^{\mathsf{Q},1}_{\mathsf{PKE}}(\lambda)=1\right]\right| \\ &= \left|\Pr\left[\mathsf{D}(\mathrm{H}^{\mathsf{A}}_{0-1}(\lambda))=1\right]-\Pr\left[\mathsf{D}(\mathrm{H}^{\mathsf{A}}_{1}(\lambda))=1\right]\right| \qquad (13)\end{aligned}$$

Game $\mathsf{Q}^{\mathrm{Enc}}(\mathsf{ek})$	Oracle $\mathrm{User}_2(i,\tilde{\sigma})$
$1^{\lambda} :\subseteq \mathsf{ek}$	**if** $\mathbf{S}_i[0] \neq \texttt{await}$ **then return** $\perp$
$(q,\mathbb{G},G,\mathsf{H}_q) \leftarrow \mathsf{Sch.Setup}(1^{\lambda})$	$(\texttt{await},\alpha_i,\beta_i,\rho_i,C_i,R_i,\nu_{s_i},\nu_{u_i}) := \mathbf{S}_i$
$key=\perp;\tau \leftarrow\$ \{0,1\}^*;\mathbf{S} := [\,]$	$\mathsf{vk}'_{\mathsf{Sch}} := ((q,\mathbb{G},G,\mathsf{H}_q),X');\tilde{\sigma}_i := \tilde{\sigma}$
$\mathsf{par} := ((q,\mathbb{G},G,\mathsf{H}_q),\mathsf{ek})$	**if** $\mathsf{Sch.Verify}(\mathsf{vk}'_{\mathsf{Sch}},(\nu_{s_i},\nu_{u_i}),\tilde{\sigma}_i)\neq 1$ **then**
$\mathsf{rwd} := 0;\mathsf{rwdmsg} := \perp$ (H_{0-1})	**return** $\perp$
$v \leftarrow \mathsf{A}^{\mathrm{Init},\mathrm{User}_0,\mathrm{User}_1,\mathrm{User}_2}(\mathsf{par},\mathsf{prds};\tau)$	$R' := R_i+\alpha_i G+\beta_i X$
$b' \leftarrow \mathsf{D}(\mathsf{par},v)$	$c_i := \mathsf{H}_q(R',X,m_i)+\beta_i$
return b'	$\varrho \leftarrow\$ \mathcal{R}_{\mathsf{PKE}};g \leftarrow\$ \{0,1\}^{\lvert\tilde{\sigma}_i\rvert};\nu_g \leftarrow\$ \mathcal{V}_u$
	$N := (\tilde{\sigma},g,\nu_{u_i},\nu_g,\nu_{s_i})$
	if $\mathsf{rwd}=0$ **then** // no rewind done yet
	set $\mathsf{rwd}=1$ **and run** $\mathsf{A}_2(\mathsf{st};\tau)$
	at most t **times**
	stop if $\mathsf{Sch.Verify}(\mathsf{vk}'_{\mathsf{Sch}},(\nu_{s_i},\nu'_{u_i}),\tilde{\sigma}'_i)\neq 0$
	in session i **or if** t **is reached**
	// at each run, output a different value ν'_{u_i}
	// from User_1 and collect the signature $\tilde{\sigma}'_i$
	// that A_2 sends to User_2
	if t **is not reached then**
	$\mathsf{rwdmsg} = (\tilde{\sigma}_i,\tilde{\sigma}'_i,\nu_{u_i},\nu'_{u_i},\nu_{s_i})$
	else halt the run with 0
	else if $\mathsf{rwdmsg}=\perp$ **then stop**
	$(\sigma^\star_0,\sigma^\star_1,\nu^\star_{01},\nu^\star_{11},\nu^\star_0) := \mathsf{rwdmsg}$ (H_{0-1})
	$N=(\sigma^\star_0,\sigma^\star_1,\nu^\star_{01},\nu^\star_{11},\nu^\star_0)$ (H_1)
	$S \leftarrow \mathrm{Enc}(N,N')$
	$\mathsf{stm} := (X,X',R_i,c_i,C_i,\varphi_i,\mathsf{ek},S)$
	$\mathsf{wtn} := (m_i,\alpha_i,\beta_i,\rho_i,\tilde{\sigma},g,\nu_{u_i},\nu_g,\nu_{s_i},\rho)$
	$\pi \leftarrow \mathsf{Niwi.Prove}(\mathsf{stm},\mathsf{wtn});\mathbf{S}_i = \texttt{closed}$
	return (c_i,π,S)

Fig. 27. Q paying against CPA security of PKE. The oracles Init, User_0 and User_1 are simulated to A as defined in game H_1 in Fig. 26. Note that Q has hardcoded the tuple of messages $(m_1,\ldots,m_{\mathsf{poly}(\lambda)})$ and predicates $(\varphi_1,\ldots,\varphi_{\mathsf{poly}(\lambda)})$ to use during the simulation of the oracles for A.

Game H_2**.** The game H_2 is identical to game H_1, except for sampling a message $\bar{m} \leftarrow\$ \{0,1\}^{|\mathcal{M}|}$ and modifying the behavior of the oracle User_2. Note that in game H_2, we sample $\bar{m}$ based on the description of $\mathcal{M}$[24], and we use this message in every session.

Specifically, in User_2, instead of computing the proof π using the witness $\mathsf{wtn} = (m_i,\alpha_i,\beta_i,\rho_i,\tilde{\sigma},g,\nu_u,\nu_g,\nu_{s_i},\varrho)$, we compute the proof using an alternative witness $\mathsf{wtn}' = (\bar{m},0,0,\varrho,\sigma^\star_0,\sigma^\star_1,\nu^\star_{01},\nu^\star_{11},\nu^\star_0,\varrho)$. By

[24] Here, we do not imply that it is possible to efficiently sample messages from the message space $\mathcal{M}$. Instead, we simply state that given the description of the message space, it is possible to sample random bits of the same length as the messages in $\mathcal{M}$. Indeed, it may be the case that $\bar{m} \notin \mathcal{M}$.

construction, this witness wtn' is valid for the statement $\mathsf{stm} = (X, X', R_i, c_i, C_i, \varphi_i, \mathsf{ek}, S)$ under the relation $\mathsf{R}_{\mathsf{PBSch}}$. This validity holds because the following conditions are satisfied:

$$S = \mathsf{PKE.Enc}(\mathsf{ek}, (\sigma_0^\star, \sigma_1^\star, \nu_{01}^\star, \nu_{11}^\star, \nu_0^\star); \varrho) \wedge \nu_{01}^\star \neq \nu_{11}^\star \wedge$$
$$1 = \mathsf{Sch.Verify}(\mathsf{vk}'_{\mathsf{Sch}}, (\nu_0^\star, \nu_{01}^\star), \sigma_0^\star) \wedge 1 = \mathsf{Sch.Verify}(\mathsf{vk}'_{\mathsf{Sch}}, (\nu_0^\star, \nu_{11}^\star), \sigma_1^\star)$$

where $\mathsf{vk}'_{\mathsf{Sch}} = ((q, \mathbb{G}, G, \mathsf{H}_q), X')$.

Reduction from WI of Niwi. We show that the advantage of a distinguisher D in distinguishing between the game H_1 and the game H_2 is bounded by the advantage of winning the WI game (Fig. 11), which is played by the adversary K, with access to the oracle Prove (Fig. 28), against the Niwi. Similarly, we can argue that if such a distinguisher exists, then there must exist a pair of tuples msgs^* and prds^*, and this pair is hardcoded in the algorithm of K.

According to the definition of WI game, K receives as input $\mathsf{par}_{\mathsf{R}}$ generated by Niwi.Rel, and according to PBSch in Fig. 3, $\mathsf{par}_{\mathsf{R}}$ is defined as $(q, \mathbb{G}, G, \mathsf{H}_q)$. Using this input K simulates the game H_1 for A, leveraging its oracle Prove. The only difference introduced by K lies in how the proof π is computed. Specifically, K first computes an alternative valid witness wtn' for the statement stm. Then, it queries its oracle Prove to compute π using the statement stm and both witnesses, wtn and wtn'. Finally, it passes the pair composed of par and the output of A to the distinguisher algorithm D and outputs the same output that the distinguisher outputs. By the construction of K, it holds that $\Pr\left[\mathrm{WI}_{\mathrm{WI}}^{\mathsf{K},0}(\lambda) = 1\right] = \Pr\left[\mathsf{D}(\mathrm{H}_1^{\mathsf{A}}(\lambda)) = 1\right]$ and also that $\Pr\left[\mathrm{WI}_{\mathrm{WI}}^{\mathsf{K},1}(\lambda) = 1\right] = \Pr\left[\mathsf{D}(\mathrm{H}_2^{\mathsf{A}}(\lambda)) = 1\right]$ and therefore:

$$\begin{aligned}\mathsf{Adv}_{\mathsf{Niwi}}^{\mathrm{WI}}(\mathsf{K}, \lambda) &:= \left|\Pr\left[\mathrm{WI}_{\mathsf{Niwi}}^{\mathsf{K},0}(\lambda) = 1\right] - \Pr\left[\mathrm{WI}_{\mathsf{Niwi}}^{\mathsf{K},1}(\lambda) = 1\right]\right| \\ &= \left|\Pr\left[\mathsf{D}(\mathrm{H}_1^{\mathsf{A}}(\lambda)) = 1\right] - \Pr\left[\mathsf{D}(\mathrm{H}_2^{\mathsf{A}}(\lambda)) = 1\right]\right| \end{aligned} \tag{14}$$

Game H_3. The game H_3 is identical to game H_2, except for sampling a message $\bar{\bar{m}} \leftarrow\!\!\$\ \{0,1\}^{|\mathcal{M}|}$ and the behavior of the oracle User_0. Specifically, in User_0, the message M, namely the plaintext encrypted in C_i, that is then returned to A, is now a fixed message, and is always the same for every played session and is defined as $M := (\bar{\bar{m}}, 0, 0)$.

Reduction from CPA-security of PKE. We show that the advantage of a distinguisher D in distinguishing between the game H_2 and the game H_3 is bounded by the advantage of winning the CPA game (Fig. 6), which is played by the adversary B (Fig. 29) against the PKE scheme.

First, note that if the adversary A outputs a view that is (computationally) different in H_2 and H_3, then there exists a distinguisher D that succeeds in distinguishing the two games. This must occur for some specific pair of tuples $\mathsf{msgs}^* := (m_1, \ldots, m_{\mathsf{poly}(\lambda)})$ and $\mathsf{prds}^* := (\varphi_1, \ldots, \varphi_{\mathsf{poly}(\lambda)})$. Conversely, if no such pair of tuples exists, then the two games are indistinguishable. Therefore, assuming that D succeeds in distinguishing the games, such a pair of tuples must necessarily exist. For this reason, we hardcode both msgs^* and prds^* into the algorithm of B, allowing it to reuse these values when simulating H_2 for A.

According to the definition of the CPA, B receives as input ek generated by PKE.KeyGen. With this, B simulates the game H_2 to A, using its oracle Enc. Namely, during a call of User_0 from A, B sets $M := (m_i, \alpha_i, \beta_i)$ and $M' := (\bar{\bar{m}}, 0, 0)$, then Q calls its encryption oracle on (M, M') and outputs the received ciphertext as part of the output of User_0. Finally, it passes the pair composed of par and the output of A to the distinguisher algorithm D and outputs the same output that the distinguisher outputs. By construction of B we have that $\Pr\left[\mathrm{CPA}_{\mathsf{PKE}}^{\mathsf{B},0}(\lambda) = 1\right] = \Pr\left[\mathsf{D}\left(\mathrm{H}_2^{\mathsf{A}}(\lambda)\right) = 1\right]$ and also that $\Pr\left[\mathrm{CPA}_{\mathsf{PKE}}^{\mathsf{B},1}(\lambda) = 1\right] = \Pr\left[\mathsf{D}\left(\mathrm{H}_3^{\mathsf{A}}(\lambda)\right) = 1\right]$ and therefore:

$$\begin{aligned}\mathsf{Adv}_{\mathsf{PKE}}^{\mathrm{CPA}}(\mathsf{B}, \lambda) &:= \left|\Pr\left[\mathrm{CPA}_{\mathsf{PKE}}^{\mathsf{B},0}(\lambda) = 1\right] - \Pr\left[\mathrm{CPA}_{\mathsf{PKE}}^{\mathsf{B},1}(\lambda) = 1\right]\right| \\ &= \left|\Pr\left[\mathsf{D}(\mathrm{H}_2^{\mathsf{A}}(\lambda)) = 1\right] - \Pr\left[\mathsf{D}(\mathrm{H}_3^{\mathsf{A}}(\lambda)) = 1\right]\right| \end{aligned} \tag{15}$$

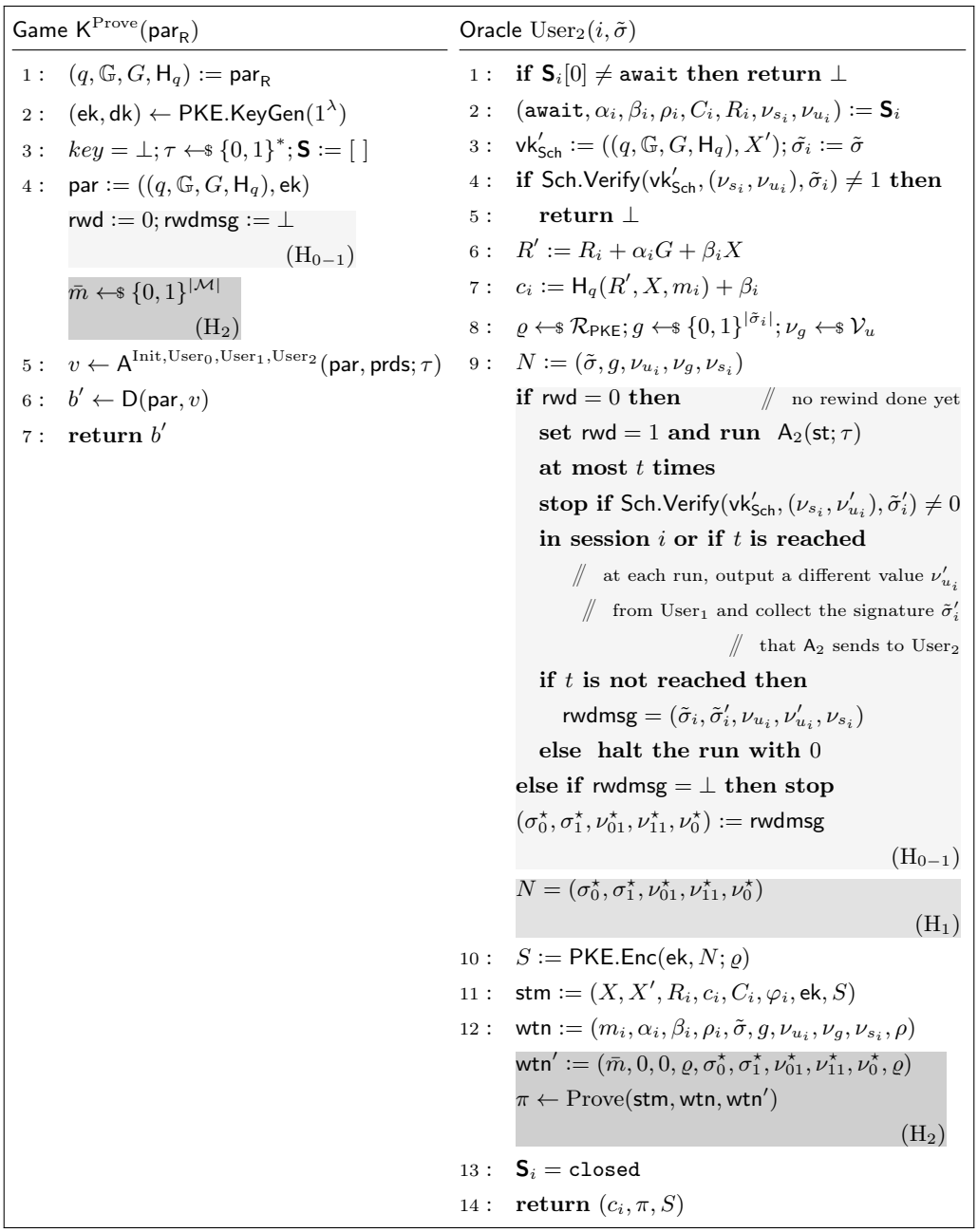


Fig. 28. K paying against WI security of Niwi. The oracles Init, User_0 and User_1 are simulated to A as defined in game H_2 in Fig. 26. Note that K has hardcoded the tuple of messages $(m_1, \ldots, m_{\mathsf{poly}(\lambda)})$ and predicates $(\varphi_1, \ldots, \varphi_{\mathsf{poly}(\lambda)})$ to use during the simulation of the oracles for A.

Game $\mathsf{B}^{\mathrm{Enc}}(\mathsf{ek})$	Oracle $\mathrm{User}_0(i)$
$1^\lambda :\subseteq \mathsf{ek}; (q, \mathbb{G}, G, \mathsf{H}_q) \leftarrow \mathsf{Sch.Setup}(1^\lambda)$	**if** $\mathbf{S}_i \neq \varepsilon \vee key = \bot$ **then**
$key = \bot; \tau \leftarrow\!\!\$\ \{0,1\}^*; \mathbf{S} := [\,]$	**return** $\bot$
$\mathsf{par} := ((q, \mathbb{G}, G, \mathsf{H}_q), \mathsf{ek})$	$\alpha_i, \beta_i \leftarrow\!\!\$\ \mathbb{Z}_q; \rho_i \leftarrow\!\!\$\ \mathcal{R}_{\mathsf{PKE}}$
$\mathsf{rwd} := 0; \mathsf{rwdmsg} := \bot$ (H_{0-1})	$M := (m_i, \alpha_i, \beta_i)$
$\bar{m} \leftarrow\!\!\$\ \{0,1\}^{\lvert\mathcal{M}\rvert}$ (H_2) $\quad \bar{m} \leftarrow\!\!\$\ \{0,1\}^{\lvert\mathcal{M}\rvert}$ (H_3)	$M' := (\bar{m}, 0, 0)$ (H_3)
$b' \leftarrow \mathsf{D}(\mathsf{par}, \mathsf{A}^{\mathrm{Init}, \mathrm{User}_0, \mathrm{User}_1, \mathrm{User}_2}(\mathsf{par}, \mathsf{prds}; \tau))$	$C_i \leftarrow \mathrm{Enc}(M, M')$
return b'	$\mathbf{S}_i = (\mathsf{open}, \alpha_i, \beta_i, \rho_i, C_i)$
	return C_i

Fig. 29. B paying against CPA security of PKE. The oracles Init, User_1 and User_2 are simulated to A as defined in game H_3 in Fig. 26. Note that B has hardcoded the tuple of messages $(m_1, \ldots, m_{\mathsf{poly}(\lambda)})$ and predicates $(\varphi_1, \ldots, \varphi_{\mathsf{poly}(\lambda)})$ to use during the simulation of the oracles for A.

Reduction from H_3 to H_S. First, consider the advantage of a distinguisher D in distinguish the game H_R and the H_S, according to the definition of deniability (Def. 4) we can write such advantage as:

$$\mathsf{Adv}^{\mathrm{H}_R, \mathrm{H}_S}_{\mathsf{PBSch}[\mathsf{P}]}(\mathsf{D}, \lambda) := \left|\Pr\left[\mathsf{D}(\mathrm{H}^{\mathsf{A}}_R(\lambda, \mathsf{msgs}, \mathsf{prds})) = 1\right] - \Pr\left[\mathsf{D}(\mathrm{H}^{\mathsf{PBS.Sim}}_S(\lambda, \mathsf{prds})) = 1\right]\right|$$

Together with the triangular inequality and Equation (13,14,15), this yields:

$$\begin{aligned}
\mathsf{Adv}^{\mathrm{H}_R, \mathrm{H}_S}_{\mathsf{PBSch}[\mathsf{P}]}(\mathsf{D}, \lambda) := \big|&\Pr\left[\mathsf{D}(\mathrm{H}^{\mathsf{A}}_R(\lambda, \mathsf{msgs}, \mathsf{prds})) = 1\right] - \Pr\left[\mathsf{D}(\mathrm{H}^{\mathsf{PBS.Sim}}_S(\lambda, \mathsf{prds})) = 1\right] + \\
&\Pr\left[\mathsf{D}(\mathrm{H}^{\mathsf{A}}_1(\lambda, \mathsf{msgs}, \mathsf{prds})) = 1\right] - \Pr\left[\mathsf{D}(\mathrm{H}^{\mathsf{A}}_1(\lambda, \mathsf{msgs}, \mathsf{prds})) = 1\right] + \\
&\Pr\left[\mathsf{D}(\mathrm{H}^{\mathsf{A}}_2(\lambda, \mathsf{msgs}, \mathsf{prds})) = 1\right] - \Pr\left[\mathsf{D}(\mathrm{H}^{\mathsf{A}}_2(\lambda, \mathsf{msgs}, \mathsf{prds})) = 1\right] + \\
&\Pr\left[\mathsf{D}(\mathrm{H}^{\mathsf{A}}_3(\lambda, \mathsf{msgs}, \mathsf{prds})) = 1\right] - \Pr\left[\mathsf{D}(\mathrm{H}^{\mathsf{A}}_3(\lambda, \mathsf{msgs}, \mathsf{prds})) = 1\right]\big| \\
&\leq \mathsf{Adv}^{\mathrm{CPA}}_{\mathsf{PKE}}(\mathsf{Q}, \lambda) + \mathsf{Adv}^{\mathrm{WI}}_{\mathsf{Niwi}}(\mathsf{K}, \lambda) + \mathsf{Adv}^{\mathrm{CPA}}_{\mathsf{PKE}}(\mathsf{B}, \lambda) \\
&\quad + \left|\Pr\left[\mathsf{D}(\mathrm{H}^{\mathsf{A}}_3(\lambda, \mathsf{msgs}, \mathsf{prds})) = 1\right] - \Pr\left[\mathsf{D}(\mathrm{H}^{\mathsf{PBS.Sim}}_S(\lambda, \mathsf{prds})) = 1\right]\right| \qquad (16)
\end{aligned}$$

Namely, that:

$$\mathsf{Adv}^{\mathrm{H}_R, \mathrm{H}_S}_{\mathsf{PBSch}[\mathsf{P}]}(\mathsf{D}, \lambda) \leq \mathsf{Adv}^{\mathrm{CPA}}_{\mathsf{PKE}}(\mathsf{Q}, \lambda) + \mathsf{Adv}^{\mathrm{WI}}_{\mathsf{Niwi}}(\mathsf{K}, \lambda) + \mathsf{Adv}^{\mathrm{CPA}}_{\mathsf{PKE}}(\mathsf{B}, \lambda) + \mathsf{Adv}^{\mathrm{H}_3, \mathrm{H}_S}_{\mathsf{PBSch}[\mathsf{P}]}(\mathsf{D}, \lambda) \qquad (17)$$

Thus, to conclude the proof, it is sufficient to show that $\mathsf{Adv}^{\mathrm{H}_3, \mathrm{H}_S}_{\mathsf{PBSch}[\mathsf{P}]}(\mathsf{D}, \lambda) := 0$ To prove this, we evaluate the output of both games. We have already discussed during the description of PBSch.Sim that the output H_S is $\bot$ only with negligible probability (i.e., when the rewinding of A made by PBSch.Sim does not succeed). Thus, except with negligible probability, in both H_3 and H_S, the output consists of:

$$(\mathsf{par}, \{X, X', \mathsf{par}, \mathsf{prds}, (C_i, R_i, \nu_{s_i}, \nu_{u_i}, \tilde{\sigma}_i, c_i, \pi_i, S_i)_{i \in [\mathsf{poly}(\lambda)]}\}).$$

By construction, for every $i \in [\mathsf{poly}(\lambda)]$, all values in this set are computed identically in both games, except for c_i. In H_3, the value c_i is computed as $c_i = \beta_i + H_i$, where β_i is sampled uniformly from $\mathbb{Z}_q$ and it remains hidden from the adversary, and $H_i = \mathsf{H}_q(R_i + \alpha_i G + \beta_i X, X, m_i)$. Conversely, in H_S, each c_i is sampled uniformly from $\mathbb{Z}_q$.

From another viewpoint, in both games, the values c_i are uniformly sampled from $\mathbb{Z}_q$. However, in H_3, an additional value is added to c_i. Despite this, the distribution of c_i remains unchanged compared to H_S because β_i is uniformly sampled from $\mathbb{Z}_q$ and never revealed to the adversary. Therefore, the outputs of H_3 and H_S are statistically identical, implying that c_i in the two games is indistinguishable to D. Consequently, we obtain:

$$\left|\Pr\left[\mathsf{D}(\mathrm{H}^{\mathsf{A}}_3(\lambda, \mathsf{msgs}, \mathsf{prds})) = 1\right] - \Pr\left[\mathsf{D}(\mathrm{H}^{\mathsf{PBS.Sim}}_S(\lambda, \mathsf{prds})) = 1\right]\right| = 0 \qquad (18)$$

The proof now follows from Equations (17,18). □

E Performances and Comparisons with [FW24]

Comparing the setup/model assumptions. Comparing our construction, PBSch, to the scheme FWSch from [FW24], we improve upon the underlying assumptions, as summarized in Table 1 and detailed in Sec. 3.2. Moreover, we obtain the deniability property. We present an instantiation of our construction that completely removes the need for a trusted setup by relying solely on the NPRO model[25]. Furthermore, we also mentioned the possibility of having a construction that requires neither a trusted setup nor reliance on the NPRO model, instead by making use of complexity leveraging, to instantiate the extractable commitment

In [FW24, Section 5.1], the authors acknowledge the criticality of relying on a trusted setup, especially because plain Schnorr signatures do not. To mitigate the need for a trusted setup for the NIZK, they propose an alternative approach where one of the two parties generates the CRS while using a "subversion" zero-knowledge proof system. This mechanism allows a party to detect whether the CRS has been tampered with. As a result, even if the verifier (i.e., the signer) sets up the CRS for the NIZK maliciously, the prover (i.e., the user) can still generate proofs without risking any leakage of the witness. However, this approach introduces significant drawbacks for the user. First, it makes their scheme, which relies on this *subversion zero-knowledge proof*, secure under a "knowledge-type" assumption. These assumptions are well known to be less standard, whereas the security of the scheme in [FW24] does not inherently depend on such assumptions[26]. Second, from a practical standpoint, in [FW24, Table 1] the authors benchmark the computational overhead introduced by these CRS checks. For a proof of a 256-bit message using the secp256k1 curve (as used in Bitcoin) and the SHA-256 hash function, the proof generation takes under a minute, whereas verifying the CRS requires approximately 4 hours. Thus, we also significantly improve upon this aspect both in terms of security issues and practicality by *completely* eliminating the need for a trusted setup, at the only price of assuming the NPRO model.

Practicality of our construction. Following [FW24] we focus on benchmarking the WI proof. The authors in [FW24] benchmarked the NArg using different types of ZK-SNARKs [BCC+17], namely Groth16 [Gro16], Plonk [GWC19], and Spartan [Set20].

Specifically, the computation of the proof π is the most challenging aspect that could render both the FWSch and our PBSch impractical. This is because (a) we cannot assume that the user has access to significant computational power, and (b) it is well known that generating a proof for the evaluation of SHA-256 is computationally demanding. Our objective is to identify whether, despite our relation $\mathsf{R}_{\mathsf{PBSch}}$ being more "complex" than the one used in [FW24], proof computation remains practical (i.e., whether it can still be carried out by an average user within a reasonable time frame). It would be unrealistic to claim that our proving time is faster than that reported for the relation used in [FW24], as our relation inherently extends the one from [FW24]. That is, we focus on benchmarking our performance in the context of PBS, particularly when using the scheme from [FW24]. Since their PBS scheme does not satisfy the deniability property, we aim to assess if our scheme, which satisfies this property, remains computationally efficient.

To ensure a fair comparison, we adopt the exact same circuit compiler as in [FW24], namely `circom` [Ide]. We use their circuits, which are designed for the relation in FWSch (see Fig. 16), and modify them to construct circuits for our relation $\mathsf{R}_{\mathsf{PBSch}}$. In our view, this approach ensures the fairest comparison since the relation in FWSch can essentially be view as part of $\mathsf{R}_{\mathsf{PBSch}}$. Finally, we compute the arithmetic complexity of the relation $\mathsf{R}_{\mathsf{PBSch}}$ and compare it with the arithmetic complexity of the relation used in FWSch[27].

In Table 2, we present the experiments we conducted following [FW24], measuring the arithmetic complexity of both relations. We opted for this measurement because, it provides a reliable understanding of the computational burden of a proof computation. This is because (a) such a value is independent of the specific hardware employed, and (b) it remains independent of the proof system used, as long as the system is based on the same arithmetization techniques.

In Table 2, we compared the arithmetic complexity of the relation used in the NArg employed in the FWSch construction with the relation used in the WI proof employed in the PBSch construction

[25] Note that, as stated by the authors in [FW24], their trusted setup (and thus also ours, as it is a reduced version) can be converted into an untrusted one by relying on the RO model. The crucial difference is that the RO allows for programmability, enabling an immediate swap from a trusted setup (via a CRS) to the RO model. This is not the case with the NPRO model, which is less powerful, closer to real-world assumptions [CJS14], and does not allow the reduction/simulator to program it.

[26] This mainly depends on the NIZK used, but many constructions rely on milder assumptions.

[27] For details on their relation and design choices of their circuits, please refer to Fig. 16.

for different application scenarios, following [FW24]. The full code used to conduct this comparison is publicly available [Rep25].

As expected, the arithmetic complexity of $\mathsf{R}_{\mathsf{PBSch}}$ is greater than that of $\mathsf{R}_{\mathsf{FWSch}}$ since the entire $\mathsf{R}_{\mathsf{FWSch}}$ is contained within $\mathsf{R}_{\mathsf{PBSch}}$. This is evident when analyzing both relations. However, the key point we want to highlight is that our construction remains practical in terms of proof computation and verification, both in terms of time and memory consumption. Specifically, the highest number of constraints in the **(A)** scenario is approximately 16 000. According to [FW24], computing around 8 000 constraints on an "average" hardware setup takes approximately less than a second (0.8 sec in their case). Thus, computing 16 000 constraints will require less than two seconds, confirming the feasibility of our approach.

The **(B)** scenario is particularly interesting. First, because it represents a real-world use case in which a Schnorr signature is employed in Bitcoin (i.e., benchmarks **(B1)** and **(B2)**). Second, the difference in constraints between the two relations is small. This is mainly because secp256k1 and SHA-256 are not SNARK-friendly choices: secp256k1 is not natively supported in `circom` like BJB, and SHA-256 is not optimized for ZK-SNARKs. Consequently, performing computations on this curve and evaluating this hash function are the most computationally intensive operations. As a result, the additional overhead introduced in $\mathsf{R}_{\mathsf{PBSch}}$ does not significantly impact efficiency. Thus, we conclude that a proof for $\mathsf{R}_{\mathsf{PBSch}}$ still remains practical to compute for an "average" user.

F Weak OMDL

We recall the weak one-more discrete logarithm (wOMDL) problem from [FW24], used in our proof of unforgeability for the predicate blind signature.

The wOMDL problem involves computing the discrete logarithm of a group element from a challenge oracle, with access to a discrete-logarithm oracle for all other elements. Unlike the OMDL game [BNPS03], the DL oracle here is restricted to challenge elements, making wOMDL a weaker assumption implied by DL.

Definition 18 (wOMDL). *Consider the game* wOMDL *in Fig. 30. A group generation algorithm* GrGen *satisfies the* weak one-more-discrete logarithm *assumption if for every* PPT *adversary* A *the advantage* $\mathsf{Adv}^{\mathrm{wOMDL}}_{\mathsf{GrGen}}(\mathsf{A},\lambda)$ *is negligible in* λ *where:*

$$\mathsf{Adv}^{\mathrm{wOMDL}}_{\mathsf{GrGen}}(\mathsf{A},\lambda) := \Pr\left[\mathrm{wOMDL}^{\mathsf{A}}_{\mathsf{GrGen}}(\lambda) = 1\right]$$

Lemma 1 ([FW24, Lemma 1]). *For every* PPT *algorithm* A *playing in* wOMDL *game that calls the* DLog *oracle* q *times, for sufficiently large* q*, there exists a* PPT *algorithm* B *playing in* DLOG *game (Fig. 5) such that:*

$$\mathsf{Adv}^{\mathrm{wOMDL}}_{\mathsf{GrGen}}(\mathsf{A},\lambda) \simeq q \cdot \exp(1) \cdot \mathsf{Adv}^{\mathrm{DLOG}}_{\mathsf{GrGen}}(\mathsf{B},\lambda) \tag{19}$$

Game $\mathrm{wOMDL}^{\mathsf{A}}_{\mathsf{GrGen}}(\lambda)$	Oracle Chal()	Oracle DLog(i)
1 : $(q,\mathbb{G},G) \leftarrow \mathsf{GrGen}(1^\lambda)$	1 : $x \leftarrow\!\!\$\ \mathbb{Z}_q$	1 : $\mathbf{R} = \mathbf{R} \| \mathbf{C}_i$
2 : $\mathbf{C} := [\,]; \mathbf{R} := [\,]$	2 : $X := xG$	2 : **return** $\mathbf{C}_i$
3 : $x^* \leftarrow \mathsf{A}^{\mathrm{Chal,DLog}}(q,\mathbb{G},G)$	3 : $\mathbf{C} = \mathbf{C} \| x$	
4 : **return** $(\lvert\mathbf{C}\rvert > 0 \wedge x^* \in \mathbf{C} \wedge x^* \notin \mathbf{R})$	4 : **return** X	

Fig. 30. The wOMDL game for a group generator GrGen played by the adversary A.

G On the Relation Among Blindness and Deniability

The main idea of the proof of Theorem 3 is as follows: starting with a generic (predicate) blind signature scheme where the output message-signature pair has high minimum entropy, we show that it is always possible to construct schemes that satisfy the blindness property (as per Def. 3) but do not satisfy deniability (according to Def. 4). Conversely, we demonstrate that it is possible to construct schemes that satisfy deniability but do not satisfy the blindness property. This separation between the two notions holds for all blind signature schemes with high minimum entropy.

To demonstrate this, we first introduce two auxiliary lemmas. Specifically, we prove both lemmas, and their results directly serve in establishing the proof of the theorem.

Lemma 2. *Let* PBS'_1 *be a generic PBS scheme. Then, there exists a scheme* PBS_1 *that satisfies blindness but does not satisfy deniability.*

Lemma 3. *Let* PBS'_2 *be a generic PBS scheme in which the output message-signature pairs have high minimum entropy. Then, there exists a scheme* PBS_2 *that satisfies deniability but does not satisfy blindness.*

G.1 Proof of Lemma 2 (Blindness does not imply deniability)

Consider a generic PBS scheme $\mathsf{PBS}'_1 = (\mathsf{PBS}'_1.\mathsf{Setup}, \mathsf{PBS}'_1.\mathsf{KeyGen}, \langle \mathsf{PBS}'_1.\mathsf{Sign}, \mathsf{PBS}'_1.\mathsf{User}\rangle$[28] $, \mathsf{PBS}'_1.\mathsf{Verify})$ that satisfies the blindness and deniability properties according to Def. 3 and Def. 4.We demonstrate that it is possible to construct a PBS scheme PBS_1 from PBS'_1 such that PBS_1 does not satisfy the deniability property as in Def. 4, but still satisfies the blindness property as in Def. 3.

First, we recall that the definition of deniability connects to a message space that might not be efficiently sampleable from the signer's point of view. The scheme PBS_1 works as follows. The interaction between the user and the signer initially follows the same protocol as PBS'_1. Specifically, if $\langle \mathsf{PBS}'_1.\mathsf{Sign}, \mathsf{PBS}'_1.\mathsf{User}\rangle$ is a three-round protocol, these rounds are executed as in PBS'_1. After this interaction, the user interacts with the signer and proves, using the four-message zero-knowledge argument from one-way functions of [BJY97][29] that the message m (used by the user in the protocol) belongs to the message space over which PBS_1 is defined.

Blindness of PBS_1**.** We first show that PBS_1 satisfies the blindness property according to Def. 3. First we recall that, by assumption, we know that PBS'_1 satisfies blindness, meaning that a malicious signer, after executing 2 successful (concurrent) sessions of PBS'_1, cannot determine (with more than negligible probability) which message-signature pair corresponds to which session.

The proof proceeds as follows, we reduce the advantage of an adversary in breaking the blindness of PBS_1, to the advantage of an adversary breaking the ZK property (Def. 11), along with the advantage in breaking the blindness of the underlying PBS'_1 scheme. Specifically, we construct a sequence of hybrid games: (1) In the first hybrid, we show that all information received by the adversary, excluding the ZK proof, does not help in distinguishing which session outputs which message-signature pair. Indeed, we have a reduction to the advantage of an adversary in breaking the blindness of the underlying PBS'_1 scheme, for which we assume that blindness holds. (2) In the second hybrid, we set up a reduction showing that if an adversary can still distinguish between sessions, this necessarily violates the ZK property. First, if an adversary can distinguish between sessions, there must exist a pair of messages that are used in both sessions where the adversary wins. The hybrid proceeds as follows: the message in the "first part" of the protocol (i.e., when the rounds of the PBS'_1 scheme are executed) remains unchanged, while the modification occurs in the four-message ZK argument. Specifically, consider a session i where the message m_b with $b \in \{0,1\}$ is used. In this new hybrid, to compute the ZK argument, we instead use the message m_{1-b}. Thus, the only difference between the previous hybrid and this one lies in the message (i.e., the witness) used in the ZK proof. Intuitively, any distinguishing advantage that an adversary gains between the two hybrids must derive from the ZK proof itself, contradicting the ZK property of the four-message ZK argument.

PBS_1 **does not satisfy deniability.** We now show that PBS_1 does not satisfy deniability according to Def. 4 by proving that a simulator for it cannot exist.

The impossibility of constructing a simulator for PBS_1 under Def. 4 follows immediately. Since PBS'_1 satisfies deniability, there exists a simulator $\mathsf{PBS}'_1.\mathsf{Sim}$ for this scheme. Even if a simulator for PBS_1 invokes $\mathsf{PBS}'_1.\mathsf{Sim}$ to simulate the initial interaction between a malicious signer and an honest user, the simulator of PBS_1 must still simulate the four-message ZK argument. However, since the simulator must work for a concurrent execution of the protocol, it is black-box and cannot program the parameters par. Simulating the ZK proof would, in essence, correspond to constructing a concurrent-secure four-message ZK argument in the plain model with black-box simulation. A seminal result by Canetti et al. [CKPR01] shows that such a construction is impossible.

[28] The notation $\langle \mathsf{PBS}'_1.\mathsf{Sign}, \mathsf{PBS}'_1.\mathsf{User}\rangle$ represents the interaction between the user and the signer. Specifically, for a three-round protocol, this denotes the sequential execution of the algorithms $\mathsf{PBS}'_1.\mathsf{User}_0$, $\mathsf{PBS}'_1.\mathsf{Sign}_0$, $\mathsf{PBS}'_1.\mathsf{User}_1$, $\mathsf{PBS}'_1.\mathsf{Sign}_1$, $\mathsf{PBS}'_1.\mathsf{User}_2$, $\mathsf{PBS}'_1.\mathsf{Sign}_2$, $\mathsf{PBS}'_1.\mathsf{User}_3$, where each algorithm processes an input message $\mathsf{msg}_{\mathsf{in}}$ and state st, producing an output message $\mathsf{msg}_{\mathsf{out}}$.

[29] Note that, the protocol from Bellare et al. [BJY97] is based on one-way functions, it is in the plain model and it is for all NP, in particular, it can be used to prove the statement considered here.

This leads to a contradiction, establishing that blindness does not necessarily imply deniability. Notably, we construct PBS_1 by adding only a four-message ZK argument (which can be based on any one-way function [BJY97]) to PBS'_1. Since the existence of a PBS scheme already implies the existence of one-way functions, this separation result is *unconditional*.

G.2 Proof of Lemma 3 (Deniability does not imply blindness)

In the proof of this lemma, we make use of strong seeded randomness extractors and the notion of high min-entropy, concepts we have so far only discussed informally. We assume the reader is familiar with these notions; for a formal treatment, we refer to Wigderson's book [Wig19, Section 9] and the references therein. In particular, we rely on the information-theoretic existence of strong seeded extractors.

Consider a generic PBS scheme $\mathsf{PBS}'_2 = (\mathsf{PBS}'_2.\mathsf{Setup}, \mathsf{PBS}'_2.\mathsf{KeyGen}, \langle \mathsf{PBS}'_2.\mathsf{Sign}, \mathsf{PBS}'_2.\mathsf{User} \rangle, \mathsf{PBS}'_2.\mathsf{Verify})$ that satisfies the blindness and deniability properties according to Def. 3 and Def. 4, and at the end the interaction between $\mathsf{PBS}'_1.\mathsf{Sign}$ and $\mathsf{PBS}'_1.\mathsf{User}$ outputs message-signature pairs with high minimum entropy (to the user). We demonstrate that it is possible to construct a PBS scheme PBS_2 from PBS'_2 such that PBS_2 does not satisfy the blindness property as in Def. 3, but still satisfies the deniability property as in Def. 4.

The scheme PBS_2 works as follows. The interaction between the user and the signer initially follows the same protocol as PBS'_2. Specifically, if $\langle \mathsf{PBS}'_2.\mathsf{Sign}, \mathsf{PBS}'_2.\mathsf{User} \rangle$ is a three-round protocol; these rounds are executed exactly as in PBS'_2. Then, an additional round is introduced: the user first computes the signature using the information obtained during the execution of the PBS'_2 rounds. Then, the user simply sends the output of a strong seeded randomness extractor[30], applied to the (high min-entropy) message-signature pair computed in the previous step. Note that the value received in this round is statistically indistinguishable from a uniformly random string and therefore carries no additional information.

Deniability for PBS_2. We now show that PBS_2 satisfies the deniability property according to Def. 4. By assumption, PBS'_2 already satisfies deniability, meaning there exists a simulator $\mathsf{PBS}'_2.\mathsf{Sim}$ that can simulate the entire interaction between a malicious signer and a user following the protocol of PBS'_2. To construct a simulator $\mathsf{PBS}_2.\mathsf{Sim}$ for PBS_2, we proceed as follows. First, we run $\mathsf{PBS}'_2.\mathsf{Sim}$, which simulates the interaction exactly as in PBS'_2. Then, $\mathsf{PBS}_2.\mathsf{Sim}$ samples a random string matching the length of the user's last message. Finally, $\mathsf{PBS}_2.\mathsf{Sim}$ outputs the output of $\mathsf{PBS}'_2.\mathsf{Sim}$ along with this random string.

The proof that the adversary's view in a real interaction with an honest user running PBS_2 is indistinguishable from the simulated interaction using $\mathsf{PBS}_2.\mathsf{Sim}$ is straightforward. The only additional information in the simulated output, compared to $\mathsf{PBS}'_2.\mathsf{Sim}$ (which is already indistinguishable from an execution with an honest user of PBS'_2), is a random string. In an actual interaction with a user of PBS_2, there is also a string that looks random. Since we already assume that the output of $\mathsf{PBS}'_2.\mathsf{Sim}$ is indistinguishable from the interaction of an honest user with a malicious signer in PBS'_2, the only remaining difference is between a truly random string in the simulated distribution and the string sent in the real execution which is clearly indistinguishable.

PBS_2 does not satisfy blindness. We now show that PBS_2 does not satisfy the blindness property as defined in Def. 3. The reason is immediate and follows directly from the fact that, in PBS_2, the message sent by the user to the signer in the last round includes the final string computed over the signature-message pair. When the message-signature pairs are later revealed, the signer can easily identify the session in which each signature was issued. This contradicts the blindness property, establishing that deniability does not necessarily imply blindness. □

G.3 Practical relevance of Thm. 3 for the construction in [FW24]

We show that the scheme introduced in [FW24], while satisfying blindness as demonstrated by the authors, does not satisfy the deniability property as in Def. 4. We highlight that this fact is significant for two main reasons. First, to our knowledge, this is the only concurrent-secure blind Schnorr signature scheme in the literature. Second, this result demonstrates that the Thm. 3 is not only valid for ad-hoc or specially crafted constructions that are designed to fail, which might be seen as having purely theoretical

[30] Note that, since a strong seeded randomness extractor ensures that the output appears uniformly random even when the seed is known, the seed can be safely sent to the signer without compromising security [Wig19, Section 9], we omit sending the seed for clarity however this would be the same in both distributions.

relevance. On the contrary, this theorem is also valid for natural, state-of-the-art constructions considered secure and practical.

Specifically, consider the scheme FWSch introduced by Fuchsbauer and Wolf [FW24], which we review in Fig. 16. According to [FW24, Theorem 2], this scheme satisfies the blindness property under the zero-knowledge property of the NArg scheme and the CPA security of the PKE. Nevertheless, we show that this scheme does not satisfy deniability.

First, recall that in both the rDNB and sDNB games from Def. 4, neither the predicates nor the messages to play in every session are chosen by the adversary, since the definition also captures the case where a signer cannot efficiently sample messages from the message space (note that, we already pointed out in Sec. 4 that this happens in several natural scenarios). Now, consider the scheme FWSch of [FW24] (Fig. 16) defined over a message space that is not efficiently sampleable by the signer (i.e., every message in the message space is generated using secret information unknown to the signer). As a concrete example, in the real world, the message space could be defined as the set of all valid signatures for a certain secret key that is not known by the signer (see Sec. 4 for an in-depth real-world example).

Suppose that the scheme FWSch also satisfies deniability. Then, according to Def. 4, there must exist an algorithm $\mathsf{FWSch.Sim}$ that play in the sDNB game (where the PBS scheme used is FWSch) and produce an output indistinguishable from that of an adversary playing in the rDNB game.

First, note that the setup algorithm $\mathsf{FWSch.Setup}$, which generates $\mathsf{par} := (\mathsf{crs}, \mathsf{ek})$ according to [FW24], is executed by a trusted party. Since the party is trusted, we are sure that it does not save the "toxic waste", namely, the trapdoor τ output during the $\mathsf{NArg.Setup}$ algorithm (along with the CRS crs), and the decryption key dk output during the $\mathsf{PKE.KeyGen}$ algorithm (along with the encryption key ek). According to deniability (see Def. 4), the simulator $\mathsf{FWSch.Sim}$ cannot locally run $\mathsf{FWSch.Setup}$ to generate its own parameters par', pass them to the adversary, and keep the corresponding tuple (τ', dk'). This is because the output of both the sDNB and rDNB games consists of the pair composed of par and (a) the simulator's output in sDNB or (b) the adversary's output in rDNB. Consequently, any distinguisher could easily distinguish between the two games if a different par' is passed to A that is not in the first entry of the pair. Thus, we must consider the case where the simulator $\mathsf{FWSch.Sim}$ attempts to simulate the adversary's execution without knowledge of the tuple (τ, dk) corresponding to par.

During such simulation, the key issue occurs when the simulator needs to simulate the interaction between the adversary and the oracle that outputs the proof π (i.e., the oracle that internally runs $\mathsf{FWSch.User}_1$ in the rDNB game). In this scenario, the simulator $\mathsf{FWSch.Sim}$ must generate an accepting proof π. Looking at the execution of $\mathsf{FWSch.User}_1$ in rDNB, we see that this proof is computed for a witness $\mathsf{wtn} := (m, \alpha, \beta, \rho)$, where m is the message to be signed. Since the simulator doesn't know a valid message m that could serve as a witness (i.e., the simulator of the sDNB game runs without any message from the message space), the simulator cannot use the $\mathsf{NArg.Prove}$ algorithm because it lacks a valid witness as input. Moreover, the simulator cannot use the $\mathsf{NArg.Sim}$ algorithm, as this would require knowledge of the trapdoor τ for the CRS, which the simulator does not hold.

Specifically, by a sequence of hybrids, we can reduce the ability of a simulator to make the output of the rDNB game indistinguishable from that of the sDNB game to an adversary breaking the soundness of the NArg. The key insight is that any accepting proof generated by the simulator must rely on either a valid witness or the trapdoor of the CRS. If the simulator could succeed without access to either, it would directly yield an adversary violating the soundness of the NArg.