(Concurrently Secure) Blind Schnorr from Schnorr

Georg Fuchsbauer and Mathias Wolf

TU Wien, Austria
`first.last@tuwien.ac.at`

22 November 2022

Abstract. Many applications of blind signatures, such as those for blockchains, require the resulting signatures to be compatible with the existing system. This makes schemes that produce Schnorr signatures, which are now supported by major cryptocurrencies, including Bitcoin, desirable. Unfortunately, the existing blind-signing protocol has been shown insecure when users can open signing sessions concurrently (Eurocrypt'21). On the other hand, only allowing sequential sessions opens the door to denial-of-service attacks.
We present the first concurrently secure blind-signing protocol for Schnorr signatures, using the standard primitives NIZK and PKE and assuming that Schnorr signatures themselves are unforgeable. We cast our scheme as a generalization of blind and partially blind signatures. We formally define the notion of predicate blind signatures, in which the signer can define a predicate that the blindly signed message must satisfy.

Keywords: Schnorr signatures, (partially) blind signatures, concurrent security

1 Introduction

Blind signatures, introduced by Chaum [Cha82], define a protocol between a signer and a user that lets the latter obtain a signature on a message hidden from the signer. Initially envisioned for e-cash systems [Cha82, CFN90, OO92, Bra94, HKOK06, BCKL09], they have also become a central primitive for e-voting protocols [Cha88, FOO93, Her97, ROG07] and anonymous credentials [Bra94, CL01, BCC$^+$09, BL13, Fuc11].

Recently, blind signatures have seen a renewed interest due to their applicability in privacy-sensitive settings ranging from *COVID-19 contact-tracing* solutions [BRS20, DLZ$^+$20] to *advanced VPNs* and *private relays* [Goo, App]. In the context of blockchains, blind signatures have been considered for increasing on-chain privacy, e.g. via *blind coin swaps* or *trustless tumbler services* [HAB$^+$17, Nic19, LLL$^+$19]. While blind-signing protocols that yield signatures of a standardized scheme are desirable in general, this is a stringent requirement in the blockchain setting, where changing the supported signature schemes requires majority consensus, which is a lengthy process.

The central scheme today are Schnorr signatures [Sch90], which also enable privacy and scalability improvements [BDN18, MPSW19, BK22]. Schnorr signatures are now supported by major blockchains such as *Bitcoin* [WNR20], *Bitcoin Cash*, *Litecoin* or *Polkadot*, and, in the form of EdDSA (and other variants), in *Monero*, *Zcash*, or *Cardano*. *Mimblewimble* [FOS19] is a cryptocurrency protocol that crucially relies on Schnorr signatures.

Since their patent expired in 2008, Schnorr signatures are outpacing RSA signatures in application counts, since for a comparable security level they are much smaller, can be verified more efficiently and are less error-prone in implementations since fewer edge cases have to be considered. While the efficiency of (EC)DSA, a NIST standard, is comparable to Schnorr, it requires non-standard assumptions to be proved secure [FKP17]. Schnorr signatures, in the form of EdDSA [BDL$^+$12], are now considered for standardization.[1]

The security of Schnorr signatures was proved under the discrete logarithm assumption (DL) [PS00] in the random oracle model (ROM) [BR93], an idealized model in which cryptographic hash functions are treated as random functions. While the proof incurs a security loss due to rewinding techniques, tight security proofs have also been given [FPS20] under DL in more idealized models such as the algebraic group model (AGM) [FKL18] together with the ROM.[2]

Blind Schnorr signatures. Schnorr signatures admit a very elegant blind-signing protocol [CP93] consisting of three messages (2 rounds). A drawback of multi-round protocols is that they might become insecure when the signer runs several signing sessions simultaneously. In applications, concurrent security is however essential, as otherwise the signer can only engage in a signing session after the previous session has been finished or canceled. Protocols that only allow sequential execution are vulnerable to denial-of-service (DoS) attacks, such as simple connection time-outs, and therefore are severely limited in throughput range. This motivated the development of concurrently secure blind signature schemes [Bol03, BNPS03, Oka06, GG14, FHS15, KLX22].

To analyze the (concurrent) security of the original blind Schnorr signing protocol [CP93], Schnorr [Sch01] introduced the so-called "ROS problem" and showed that in the generic group model [Nec94, Sho97] together with the ROM, and assuming ROS was hard, blind Schnorr signatures were unforgeable. Conversely, Schnorr also showed that solving the ROS problem enables an attack on the scheme when the adversary can engage in concurrent signing sessions. Wagner's subexponential-time attack [Wag02] showed that ROS was not as hard as conjectured, and recently Benhamouda et al. [BLL$^+$21] presented a polynomial-time algorithm. They show how an attacker that opens polynomially many signing sessions (concretely, 256, if that is the security parameter) can efficiently forge signatures (concretely, derive 257 signatures on messages of his choice).

Earlier, Fuchsbauer, Plouviez and Seurin [FPS20] had proposed a variant for blind Schnorr signing that does not succumb to the ROS attack. In their *clause blind Schnorr* scheme, the signer and user engage in two parallel Schnorr blind-signing sessions of which the signer randomly finishes just one. They prove unforgeability in the algebraic group model and the ROM from the *one-more discrete logarithm* (OMDL) assumption [BNPS03] (which holds in the generic group model [BFP21]) and the assumption that a new "modified ROS problem" is infeasible; an assumption that appears weaker than ROS but remains unstudied. The question whether the original blind Schnorr signature scheme is secure when signing sessions are only performed sequentially was answered in the affirmative by Kastner, Loss and Xu [KLX22], who give a proof from OMDL in the AGM+ROM.

The lack of concurrently secure blind signing protocols for Schnorr signatures with solid security guarantees has led to today's unsatisfactory situation: while Schnorr signatures are replacing RSA signatures, the ongoing standardization effort by the IETF [DJW22] for blind signatures only specifies RSA blind signatures [Cha82], which they prefer over *clause blind*

[1] https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-5-draft.pdf

[2] The AGM assumes the adversary against a cryptosystem defined over a group $(\mathbb{G}, +)$ to be *algebraic*, which means that if, after having received group elements $X_1, \ldots, X_n$, the adversary returns a group element Z, one can extract a *representation* $(\zeta_1, \ldots, \zeta_n)$ of Z, that is, $Z = \sum \zeta_i X_i$.

Schnorr [FPS20] – despite RSA having much larger key and signature sizes (and not lending themselves as nicely to evaluation batching or efficient threshold signing as (blind) Schnorr signatures, as the authors concede [DJW22]).

Partially blind signatures. Blind Schnorr signatures, as well as most of the mentioned schemes, only provide "full" blindness, meaning the signer learns nothing about the message she is signing (and she cannot link the signature to the signing session it was produced in). In practice, this is often too strong and it is preferable to only hide parts of the signed message, while giving the signer control over the remaining parts. This is what "partially" blind signatures, introduced by Abe and Okamoto [AO00], provide. In their model, a message consists of a public and a secret part and the signer gets to see the former during signing.

Our contributions. We present the first concurrently secure blind signing protocol for producing Schnorr signatures with rigorous security guarantees. Our starting point is the plain protocol [CP93], against which the recent attack [BLL^{+}21] proceeds as follows. The adversary, impersonating the user, opens λ many signing sessions, where λ is the bit length of the order of the underlying group. The adversary thus obtains the signer's first protocol message, a group element R, for each session. For every session it then samples two possible sets of "blinding values" (these represent the randomness used by the user during a session). The R values obtained from the signer then determine which set of blinding values the attacker will use in every session in order to compute the forgeries. The crucial observation is that before receiving R, the attacker does not know which blinding values it will use.

A first attempt to prevent this attack could be to oblige the user to commit to her blinding values (as well as the message to be signed) before receiving the value R. In the second round, the user then proves that her next protocol message is consistent with the committed values; she does so using a zero-knowledge proof. Only if the proof verifies will the signer send the last message, which lets the user compute the signature. Somewhat surprisingly, this modification suffices to not only defend against this particular attack, but to make the scheme unforgeable under concurrent signing sessions, for which we give a security proof.

We also observe that when the user sends a proof that her protocol message is consistent with the (committed) message to be signed, she might as well prove any property about this message. Our construction therefore instantiates a more general primitive than blind, and even partially blind, signatures, which we formally define (see below).

Concretely, our construction uses a public-key encryption scheme PKE for the "commitment" in the first round[3] and a non-interactive zero-knowledge (NIZK) argument system NArg [BFM88, BCC88] for the proof in the second round. While the plain protocol [CP93] is unconditionally blind, we show that our construction satisfies a computational notion assuming that NArg is zero-knowledge and PKE satisfies the standard notion of chosen-plaintext security. We prove unforgeability of our construction assuming that NArg is sound and that Schnorr signatures themselves are secure for the underlying group and hash function (families). While the latter might seem like an unusual assumption, it is arguably a minimum assumption in any scenario that uses Schnorr signatures. The reason we explicitly require it is that the statement proved by the NIZK scheme involves the concrete hash function used by the signature scheme, so we cannot rely on the security of Schnorr signatures in the random oracle model.

[3] The reason is that our security proof requires extracting the committed values and we do not want to rely on strong (possibly non-blackbox) extraction assumptions for commitments; hence we simply use a PKE.

Avoiding a trusted setup. Our security notions and proofs assume trusted parameters (which is necessary for NIZKs in the standard model [GO94]), but depending on the instantiation of NArg and PKE, a trusted setup can easily be avoided in practice: Instantiating PKE e.g. with ElGamal over an elliptic-curve group, one could generate a public key for which no one knows the secret key by "hashing into the curve" [BF01, BCI+10] (formally, this would be proved secure in the ROM). For the NIZK, one can also use schemes that are secure in the ROM or require a "uniform reference string" [BBB+18, BCR+19, BFS19, BGH19, KPV19, BBHR19, COS20, Set20, CHM+20, SL20] (which could also be created via a hash function).

If the signer sets up the NIZK parameters (and can thus be sure that no one knows a potential simulation trapdoor), more efficient schemes can be used if they are subversion-zero-knowledge [BFS16]; that is, they remain ZK even under adversarially generated parameters. (Blindness of the scheme, which protects against malicious signers, would then still hold.) For example, *Groth16* [Gro16], the zk-SNARK with the shortest proofs, has been shown to satisfy this notion [Fuc18] when the prover first performs a consistency check on the NIZK parameters. When engaging with the signer, users would have to perform this (potentially complex) check, but in practice, one could optimistically trust the signer, since someone discovering that her parameters are not consistent would harm her reputation.

Another possibility is to accept trusted parameters, but use a scheme that has "universal" parameters [GKM+18], such as [GWC19, CHM+20]. These parameters need only be generated in a trusted way once and can then be used to prove any statement (up to a certain size). In terms of efficiency, the main bottleneck of our construction is the NIZK proof system, for which however any of the nowadays high-performance zk-SNARK constructions mentioned so far can be used. We discuss implementation details and the resulting transparency and efficiency in Section 5.

EdDSA. Since EdDSA [BDL+12] is based on Schnorr signatures, our construction also yields (predicate) blind EdDSA signatures. EdDSA, as well as Schnorr signatures in practice [WNR20] derive the randomness r used during signing in a deterministic way, by hashing the message to be signed together with the signer's secret key. (This prevents leakage of the secret key when the same r is used for different messages and makes the scheme more resilient to side-channel attacks [NS02].) This method is not applicable during blind signing, as the signer does not know the message. A blind signature would thus be distributed differently to a (derandomized) standard signature, but computationally indistinguishable.

Generalizing blind signatures. We introduce the notion of predicate blind signatures (PBS), which generalizes the concept of partially blind signatures [AO00] and improves on the privacy guarantees. While in partially blind signatures, the signer and the user agree on the public part of the message to be signed before engaging in the signing protocol, in PBS they agree on a predicate on the message to be signed. After successful completion, the signer is guaranteed that the message she signed satisfies the predicate and the user is guaranteed that the signer learned nothing more than that. In addition, the signature does not reveal anything about the predicate. This is in contrast to partially blind signatures, for which the public (i.e., agreed upon) part is part of the message.

Predicate blind signatures address problems of conditional authorization in a general way. For example, a bank may only authorize customer transactions that are compliant with the law and/or internal rules, and at the same time aim at protecting customer privacy. E.g., it could be required that a transaction does not exceed a certain amount if its recipient is located in a list of declared countries. Using PBS, the bank would not learn the amount nor the recipient's location, but that the criteria are met.

Trying to realize such scenarios with partially blind signatures would require stating the conditions explicitly in the public message part. This would make signature verification more cumbersome, since it is left to the verifier to check whether the rest of the message conforms to the public part. Worse, the signature would reveal the bank's policy, which remains hidden when using PBS. We believe that the concept of predicate blind signatures better suits real-world demands than currently existing variants of blind signatures. Our construction gives a (concurrently secure) instantiation of PBS whose signatures are standard Schnorr signatures.

2 Preliminaries

We introduce the notation used throughout this work and recall the primitives upon which our predicate blind Schnorr signature protocol, given in Section 4, builds, as well as their security notions. As the primitives are quite standard, we present them without detailed elaboration.

2.1 Notation

For $n \in \mathbb{N}^+$ we denote by $[n]$ the set $\{1, \dots, n\}$. We let $a \coloneqq b$ 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 variable b's value. Or, if it is clear from the context, denotes the boolean comparison between a and b.

An empty list is initialized via $\vec{a} \coloneqq [\,]$. A value x is appended to list $\vec{a}$ via $\vec{a} = \vec{a} \| x$. The size of $\vec{a}$ is denoted by $|\vec{a}|$. We denote the j-th element of $\vec{a}$ by $\vec{a}_j$. Attempts to access a position $j \notin [|\vec{a}|]$ returns the empty symbol ε. Tuples of elements are denoted as $x \coloneqq (a, \dots, z)$ 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. $\mathcal{A}, \mathcal{B}, \mathcal{C}$, and algorithms by Sans Serif typestyle. Algorithms are considered to be efficient, i.e., run in probabilistic polynomial time (**p.p.t.**) w.r.t. to the security parameter 1^λ, which we usually keep as an implicit input. All adversaries are assumed to be efficient algorithms. For a p.p.t. algorithm X with explicit randomness r we write $y \coloneqq \mathsf{X}(x; r)$ to denote assignment of X's result on input x with randomness r to variable y. We write $y \leftarrow \mathsf{X}(x)$ for sampling r uniformly at random and assigning $y \coloneqq \mathsf{X}(x; r)$.

A function $\epsilon \colon \mathbb{N} \to \mathbb{R}^+$ is negligible if for every $c > 0$ there exists k_0 s.t. $\epsilon(k) < 1/k^c$ for all $k \geq k_0$. We assume that uniform sampling from $\mathbb{Z}_n$ is possible for any $n \in \mathbb{N}$. We let $a \leftarrow_{\$} \mathcal{A}$ denote sampling the variable a uniformly from the set $\mathcal{A}$. To enhance readability of pseudocode, if a value a "implicitly defines" values $b_1, b_2, \dots$ (that is, these can be parsed or obtained from a in polynomial time), we write $(b_1, b_2, \dots) :\subseteq a$. We write $a \equiv_p b$ to denote $a \equiv b \pmod{p}$.

2.2 Discrete-Logarithm-Hard Groups

Definition 1. *A **group generation algorithm** GrGen is a p.p.t. algorithm that takes as input a security parameter λ in unary and returns $(p, \mathbb{G}, G)$, where $\mathbb{G}$ is the description of a group of prime order p s.t. $\lceil \log_2(p) \rceil = \lambda$, and G is a generator of $\mathbb{G}$.*

Definition 2. *A group generation algorithm* GrGen *is* ***discrete-logarithm-hard*** *if for every adversary (recall that these are assumed to be p.p.t. in* λ*)* A *the function*

$$\mathsf{Adv}^{\mathsf{DL}}_{\mathsf{GrGen},\mathsf{A}}(\lambda) \coloneqq \Pr\left[\mathsf{DL}^{\mathsf{A}}_{\mathsf{GrGen}}(\lambda)\right]$$

is negligible in λ*, where game* DL *is defined by:*

```
$\mathsf{DL}^{\mathsf{A}}_{\mathsf{GrGen}}(\lambda)$
$(p, \mathbb{G}, G) \leftarrow \mathsf{GrGen}(1^\lambda)$
$x \leftarrow\!\!\$\, \mathbb{Z}_p\,;\ X \coloneqq xG$
$y \leftarrow \mathsf{A}(p, \mathbb{G}, G, X)$
return $(y = x)$
```

2.3 Non-Interactive Zero-Knowledge Arguments

We define non-interactive zero-knowledge argument (NIZK) systems with respect to parameterized relations $\mathsf{R}\colon \{0,1\}^* \times \{0,1\}^* \times \{0,1\}^* \to \{0,1\}$. We consider ternary relations that run in polynomial-time w.r.t. the first argument, the parameters, denoted par_{R}. Given par_{R}, for a statement θ we call w a witness if $\mathsf{R}(par_{\mathsf{R}}, \theta, w) = 1$, and define the language $\mathcal{L}_{par_{\mathsf{R}}} = \{\theta \mid \exists w : \mathsf{R}(par_{\mathsf{R}}, \theta, w) = 1\}$. A NIZK for a relation R is a tuple of efficient algorithms $\mathsf{NArg}[\mathsf{R}] = (\mathsf{Rel}, \mathsf{Setup}, \mathsf{Prove}, \mathsf{Vfy}, \mathsf{SimProve})$ with the following syntax:

– $\mathsf{Rel}(1^\lambda) \to par_{\mathsf{R}}$: the p.p.t. relation parameter generation algorithm takes as input the security parameter λ in unary and returns the relation parameters par_{R} s.t. $1^\lambda :\subseteq par_{\mathsf{R}}$ (i.e., 1^λ can be efficiently obtained from par_{R}) and $\mathcal{L}_{par_{\mathsf{R}}}$ is a NP-language.
– $\mathsf{Setup}(par_{\mathsf{R}}) \to (crs, \tau)$: the p.p.t. setup algorithm takes as input relation parameters par_{R} and returns a common reference string crs and a simulation trapdoor τ. The common reference string contains the description of par_{R}, i.e. $par_{\mathsf{R}} :\subseteq crs$.
– $\mathsf{Prove}(crs, \theta, w) \to \pi$: the p.p.t. prover algorithm takes as input a common reference string crs, a statement θ and a witness w and returns a proof π.
– $\mathsf{Vfy}(crs, \theta, \pi) =: 0/1$: the deterministic p.t. verification algorithm takes as input a common reference string crs, a statement θ and a proof π and outputs 1 (accept) or 0 (reject).
– $\mathsf{SimProve}(crs, \tau, \theta) \to \pi$: the p.p.t. simulation algorithm takes as input a common reference string crs, a simulation trapdoor τ and a statement θ and returns a proof π.

Definition 3. *A system* $\mathsf{NArg}[\mathsf{R}]$ *has perfect correctness if for every adversary* A *and* $\lambda \in \mathbb{N}$*:*

$$\Pr\left[\begin{array}{l} par_{\mathsf{R}} \leftarrow \mathsf{Rel}(1^\lambda) \\ (crs, \tau) \leftarrow \mathsf{Setup}(par_{\mathsf{R}}) \\ (\theta, w) \leftarrow \mathsf{A}(crs) \\ \pi \leftarrow \mathsf{Prove}(crs, \theta, w) \end{array} : \mathsf{R}(par_{\mathsf{R}}, \theta, w) = 0 \vee \mathsf{Vfy}(crs, \theta, \pi) = 1\right] = 1 .$$

Definition 4. *A system* $\mathsf{NArg}[\mathsf{R}]$ *is* ***(adaptively) computationally sound*** *if for every adversary* A

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

is negligible in λ*, where game* SND *is defined by:*

```
$\mathsf{SND}^{\mathsf{A}}_{\mathsf{NArg[R]}}(\lambda)$
$par_{\mathsf{R}} \leftarrow \mathsf{NArg.Rel}(1^\lambda)$
$(crs, \tau) \leftarrow \mathsf{NArg.Setup}(par_{\mathsf{R}})$
$(\theta, \pi) \leftarrow \mathsf{A}(crs)$
return $\big(\mathsf{NArg.Vfy}(crs, \theta, \pi) = 1\big) \wedge \big(\forall w \in \{0,1\}^* : \mathsf{R}(par_{\mathsf{R}}, \theta, w) = 0\big)$
```

Definition 5. *A system* NArg[R] *is* ***computationally zero-knowledge*** *if for every adversary* A

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

is negligible in λ*, where game* ZK *is defined by:*

```
$\mathsf{ZK}^{\mathsf{A},b}_{\mathsf{NArg[R]}}(\lambda)$
$par_{\mathsf{R}} \leftarrow \mathsf{NArg.Rel}(1^\lambda)$
$(crs, \tau) \leftarrow \mathsf{NArg.Setup}(par_{\mathsf{R}})$
$b' \leftarrow \mathsf{A}^{\textsc{Prove}}(crs)$
return $b'$

$\textsc{Prove}(\theta, w)$
if $\mathsf{R}(par_{\mathsf{R}}, \theta, w) = 0$ then return $\perp$
$\pi_0 \leftarrow \mathsf{NArg.Prove}(crs, \theta, w)$
$\pi_1 \leftarrow \mathsf{NArg.SimProve}(crs, \tau, \theta)$
return $\pi_b$
```

2.4 Public-Key Encryption

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

- $\mathsf{KeyGen}(1^\lambda) \to (ek, dk)$ is a p.p.t. algorithm that takes the security parameter in unary and outputs a public encryption key ek and a secret decryption key dk. The public key defines the message space $\mathcal{M}_{ek}$, the randomness space $\mathcal{R}_{ek}$ and the ciphertext space $\mathcal{C}_{ek}$.
- $\mathsf{Enc}(ek, M; \rho) =: C$ is a p.p.t. algorithm that takes as input a public encryption key ek, a message M, randomness $\rho \in \mathcal{R}_{ek}$, and outputs a ciphertext $C \in \mathcal{C}_{ek}$ if $M \in \mathcal{M}_{ek}$ and $\perp$ otherwise.
- $\mathsf{Dec}(dk, C) =: M$ is a deterministic p.t. algorithm that on input a ciphertext $C \in \mathcal{C}_{ek}$ and the decryption key dk outputs a message $M \in \mathcal{M}_{ek}$.

Definition 6. *A public-key encryption scheme* PKE *has* ***(perfect) correctness*** *if for every adversary* A *and* $\lambda \in \mathbb{N}$*:*

$$\Pr\left[\begin{array}{l} (ek, dk) \leftarrow \mathsf{PKE.KeyGen}(1^\lambda) \\ M \leftarrow \mathsf{A}(ek, dk) \\ C \leftarrow \mathsf{PKE.Enc}(ek, M) \end{array} : M \notin \mathcal{M}_{ek} \vee \mathsf{PKE.Dec}(dk, C) = M\right] = 1 \;.$$

Definition 7. *A public-key encryption scheme* PKE *is* ***secure against chosen-plaintext attacks*** *(CPA-secure) if for all adversaries* A

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

is negligible in λ*, where game* CPA *is defined as:*

```
$\mathsf{CPA}^{\mathsf{A},b}_{\mathsf{PKE}}(\lambda)$
$(ek, dk) \leftarrow \mathsf{PKE.KeyGen}(1^\lambda)$
$b' \leftarrow \mathsf{A}^{\textsc{Enc}}(ek)$
return $(b = b')$

$\textsc{Enc}(M_0, M_1)$
$C \leftarrow \mathsf{PKE.Enc}(ek, M_b)$
return $C$
```

2.5 Signature Schemes

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

- $\mathsf{Setup}(1^\lambda) \to sp$: the setup algorithm takes as input the security parameter λ in unary and outputs (signature) parameters sp, which define the message space $\mathcal{M}_{sp}$.
- $\mathsf{KeyGen}(sp) \to (sk, vk)$: the key generation algorithm takes parameters sp and outputs a signing key sk and a verification key vk.
- $\mathsf{Sign}(sk, m) \to \sigma$: the signing algorithm takes as input a signing key sk and a message $m \in \mathcal{M}_{sp}$ and outputs a signature σ.
- $\mathsf{Ver}(vk, m, \sigma) =: 0/1$: the (deterministic) verification algorithm takes a verification key vk, a message m and a signature σ; it returns 1 if σ is valid and 0 otherwise.

Definition 8. *A signature scheme* SIG *has* ***(perfect) correctness*** *if for every adversary* A *and* $\lambda \in \mathbb{N}$*:*

$$\Pr\left[\begin{array}{l} sp \leftarrow \mathsf{SIG.Setup}(1^\lambda) \\ (sk, vk) \leftarrow \mathsf{SIG.KeyGen}(sp) \\ m \leftarrow \mathsf{A}(sk, vk) \\ \sigma \leftarrow \mathsf{SIG.Sign}(sk, m) \end{array} : m \notin \mathcal{M}_{sp} \vee \mathsf{SIG.Ver}(vk, m, \sigma) = 1\right] = 1 \ .$$

Definition 9. *A signature scheme* SIG *satisfies* ***strong existential unforgeability under chosen-message attacks*** *(sEUF-CMA) if for all adversaries* A

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

is negligible in λ*, where game* $\mathsf{sEUF\text{-}CMA}$ *is defined by:*

```
$\mathsf{sEUF\text{-}CMA}^{\mathsf{A}}_{\mathsf{SIG}}(\lambda)$
$sp \leftarrow \mathsf{SIG.Setup}(1^\lambda)$
$(sk, vk) \leftarrow \mathsf{SIG.KeyGen}(sp)\,;\ \mathcal{Q} := \emptyset$
$(m^*, \sigma^*) \leftarrow \mathsf{A}^{\textsc{Sign}}(vk)$
return $\big((m^*, \sigma^*) \notin \mathcal{Q} \wedge \mathsf{SIG.Ver}(vk, m^*, \sigma^*) = 1\big)$

$\textsc{Sign}(m)$
$\sigma \leftarrow \mathsf{SIG.Sign}(sk, m)$
$\mathcal{Q} = \mathcal{Q} \cup \{(m, \sigma)\}$
return $\sigma$
```

2.6 Schnorr Signatures

The Schnorr signature scheme is defined w.r.t. a group generation algorithm (Definition 1) returning a group of prime order p, and it requires a hash function that maps into $\mathbb{Z}_p$, which we define as being generated as follows.

```
$\mathsf{Sch.Setup}(1^\lambda)$
    $(p, \mathbb{G}, G) \leftarrow \mathsf{GrGen}(1^\lambda)$
    $\mathsf{H} \leftarrow \mathsf{HGen}(p)$
    $sp := (p, \mathbb{G}, G, \mathsf{H})$
    return $sp$

$\mathsf{Sch.KeyGen}(sp)$
    $(p, \mathbb{G}, G, \mathsf{H}) := sp$
    $x \leftarrow\$ \mathbb{Z}_p\,;\ X := xG$
    $sk := (sp, x)\,;\ vk := (sp, X)$
    return $(sk, vk)$

$\mathsf{Sch.Sign}(sk, m)$
    $(p, \mathbb{G}, G, \mathsf{H}, x) := sk\,;\ r \leftarrow\$ \mathbb{Z}_p\,;\ R := rG$
    $c := \mathsf{H}(R, xG, m)\,;\ s := r + cx \bmod p$
    $\sigma := (R, s)$
    return $\sigma$

$\mathsf{Sch.Ver}(vk, m, \sigma)$
    $(p, \mathbb{G}, G, \mathsf{H}, X) := vk$
    $(R, s) := \sigma$
    $c := \mathsf{H}(R, X, m)$
    return $(sG = R + cX)$
```

Fig. 1. The Schnorr signature scheme $\mathsf{Sch}[\mathsf{GrGen}, \mathsf{HGen}]$ (with key prefixing, meaning that the public key $X = xG$ is part of the hash function input) based on a group generator GrGen and hash generator HGen.

Definition 10. *A (target-range)* ***hash function generator*** HGen *is a p.p.t. algorithm that takes as input a number* $n \in \mathbb{N}^+$ *and returns the description of a function* $\mathsf{H}\colon \{0,1\}^* \to \mathbb{Z}_n$.

In Figure 1 we define Schnorr signatures with "key-prefixing" [BDL+12], which is the variant in use today. (Key-prefixing protects against certain *related-key attacks* [MSM+16] and implies tight security in the multi-user setting [Ber15].) Unforgeability of Schnorr signatures has been studied extensively in the random oracle model (ROM) [BR93, PS96, PS00] and more recently in the algebraic group model (AGM) and the ROM [FPS20], which enables tight security. These proofs are easily adapted to the key-prefixing variant. Strong unforgeability of this variant (in the AGM+ROM) readily follows from the discrete-logarithm assumption and the fact that key-prefixing Schnorr signatures are *strongly simulation-extractable proofs of knowledge* of discrete logarithms in the AGM+ROM [FO22].

We consider these results and the fact that, despite being thus widely used, no vulnerabilities have been found as ample evidence for the following assumption, which we use in the security proof of our predicate blind Schnorr signature scheme:

Assumption 1. *There exists a group generator* GrGen *and a hash function generator* HGen *s.t. the Schnorr signature scheme (Figure 1) is strongly unforgeable (Definition 9), in particular, for all adversaries* A*, the function* $\mathsf{Adv}^{\mathsf{sEUF\text{-}CMA}}_{\mathsf{Sch}[\mathsf{GrGen},\mathsf{HGen}],\mathsf{A}}(\lambda)$ *is negligible in* λ.

3 Predicate Blind Signatures

We introduce predicate blind signatures (PBS) as a generalization of partially blind signatures [AO00]. PBS define an interactive protocol that enable a signer to sign a message at the behest of another party, called the user, without learning any information about the signed message, besides that it satisfies certain conditions (defined by a predicate) on which the user and signer agreed before the interaction.

A PBS scheme is parameterized by a family of polynomial-time-computable predicates, which are implemented by a p.t. algorithm P, which we refer to as a *predicate compiler*. On input a predicate description $prd \in \{0,1\}^*$ and a message $m \in \{0,1\}^*$, P returns 1 or 0 indicating whether m satisfies the predicate prd or not. A predicate blind signature scheme $\mathsf{PBS}[\mathsf{P}]$ for P is defined by the following algorithms. We focus on schemes with a 2-round (i.e., 4-message) signing protocol for concreteness.

– $\mathsf{PBS.Setup}(1^\lambda) \to par$: the setup algorithm takes as input the security parameter in unary and returns public parameters par, which define the message space $\mathcal{M}_{par}$.
– $\mathsf{PBS.KeyGen}(par) \to (sk, vk)$: the key generation algorithm takes the parameters par and returns a signing/verification key pair (sk, vk), which both implicitly contain par, i.e., $par :\subseteq vk$.
– $\langle \mathsf{PBS.Sign}(sk, prd), \mathsf{PBS.User}(vk, prd, m)\rangle \to (b, \sigma)$: an interactive protocol with shared input par (implicitly contained in sk and vk) and a predicate prd is run between the signer and user. The signer takes the secret key sk as private input, the user's private input is a verification key vk and a message m. The signer outputs $b = 1$ if the interaction completes successfully and $b = 0$ otherwise, while the user outputs a signature σ if it terminates correctly, and $\perp$ otherwise. For a 2-round protocol the interaction can be realized by the following algorithms:

$$\begin{aligned}(msg_{U,0}, state_{U,0}) &\leftarrow \mathsf{PBS.User}_0(vk, prd, m)\\ (msg_{S,1}, state_S) &\leftarrow \mathsf{PBS.Sign}_1(sk, prd, msg_{U,0})\\ (msg_{U,1}, state_{U,1}) &\leftarrow \mathsf{PBS.User}_1(state_{U,0}, msg_{S,1})\\ (msg_{S,2}, b) &\leftarrow \mathsf{PBS.Sign}_2(state_S, msg_{U,1})\\ \sigma &\leftarrow \mathsf{PBS.User}_2(state_{U,1}, msg_{S,2})\end{aligned}$$

We write $\langle \mathsf{PBS.Sign}(sk, prd), \mathsf{PBS.User}(vk, prd, m)\rangle \to (b, \sigma)$ as a shorthand for the above sequence.
– $\mathsf{PBS.Ver}(vk, m, \sigma) =: 0/1$: the (deterministic) verification algorithm takes a verification key vk, a message m and a signature σ, and returns 1 if σ is valid on m under vk and 0 otherwise.

Similar to the security definition for blind signatures [JLO97] and partially blind signatures [AO00], we require the properties we define in the following.

Definition 11. *A predicate blind signature scheme* PBS *for predicate compiler* P *is* ***(perfectly) correct*** *if for any adversary* A *and* $\lambda \in \mathbb{N}$*:*

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

(Strong) unforgeability. The security notion for blind signatures states that after the completion of n signing sessions, the user cannot compute $n+1$ signatures. Similarly, partially blind signatures require that after the completion of any number of signing sessions, of which n share the same public message part, the user cannot compute $n+1$ signatures for this public message part.

```
$\mathsf{UNF}^{\mathsf{A}}_{\mathsf{PBS[P]}}(\lambda)$
$par \leftarrow \mathsf{PBS.Setup}(1^\lambda)$
$(sk, vk) \leftarrow \mathsf{PBS.KeyGen}(par)$
$\vec{\mathrm{S}} := [\,]$        // list holding all session details
$\vec{\mathrm{PRD}} := [\,]$    // predicates of successful sessions
$(m_i^*, \sigma_i^*)_{i \in [n]} \leftarrow \mathsf{A}^{\textsc{Sign}_1, \textsc{Sign}_2}(vk)$
return $\big(n > 0$
    $\wedge\, \forall\, i \in [n]\colon \mathsf{PBS.Ver}(vk, m_i^*, \sigma_i^*) = 1$
    $\wedge\, \forall\, i \neq j \in [n]\colon (m_i^*, \sigma_i^*) \neq (m_j^*, \sigma_j^*)$
    $\wedge\, \nexists\, \rho \in \mathsf{InjF}([n], [|\vec{\mathrm{PRD}}|])\colon$
          $\forall\, i \in [n]\colon \mathsf{P}(\vec{\mathrm{PRD}}_{\rho(i)}, m_i^*) = 1\big)$
      // there is no mapping of messages to
      // predicates for successful sessions
```

```
$\textsc{Sign}_1(prd, msg)$
$(msg', state) \leftarrow \mathsf{PBS.Sign}_1(sk, prd, msg)$
$\vec{\mathrm{S}} = \vec{\mathrm{S}} \| (state, prd)$     // store new session
return $msg'$
```

```
$\textsc{Sign}_2(j, msg)$
if $\vec{\mathrm{S}}_j = \varepsilon$ then          // $j$-th session not open
    return $\bot$
$(state, prd) := \vec{\mathrm{S}}_j$
$(msg', b) \leftarrow \mathsf{PBS.Sign}_2(state, msg)$
if $b = 1$ then
    $\vec{\mathrm{S}}_j := \varepsilon$                // close session $j$
    $\vec{\mathrm{PRD}} = \vec{\mathrm{PRD}} \| prd$   // store predicate $prd$
return $msg'$
```

Fig. 2. The strong unforgeability game for a predicate blind signature scheme $\mathsf{PBS[P]}$ with a 2-round signing protocol. $\mathsf{InjF}(\mathcal{A}, \mathcal{B})$ denotes the set of all injective functions from set $\mathcal{A}$ to set $\mathcal{B}$. (Standard) unforgeability is obtained by replacing the winning condition $\forall\, i \neq j \in [n] : (m_i^*, \sigma_i^*) \neq (m_j^*, \sigma_j^*)$ with $\forall\, i \neq j \in [n] : m_i^* \neq m_j^*$.

Generalizing this to predicate blind signatures is not straightforward as messages can satisfy many predicates (whereas for partially blind signatures, messages only have one public part). We therefore simply require that anything the user can output after running signing sessions for predicates of its choice can be "explained". That is, when the user outputs signed messages $m_1^*, \ldots, m_n^*$, then there must exist a mapping to successful signing sessions, so that the message satisfies the used predicate. In particular, there must be an injective mapping $\rho\colon [n] \to [\ell]$ where ℓ is the number of closed signing sessions, so that $\mathsf{P}(prd_{\rho(i)}, m_i^*) = 1$, where prd_j was the predicate for the j-th successfully closed session.

Our notion is in the spirit of *strong* unforgeability as we consider the pairs of messages and signatures to be distinct. It also gives strong guarantees in the spirit of [FPS20], in that it only considers closed signing sessions and ignores unfinished sessions when checking whether the attack was not trivial.

Definition 12. *A predicate blind signature scheme* $\mathsf{PBS[P]}$ *satisfies **(strong) unforgeability** if for all adversaries* A

$$\mathsf{Adv}^{\mathsf{UNF}}_{\mathsf{PBS,A}}(\lambda) := \Pr\big[\mathsf{UNF}^{\mathsf{A}}_{\mathsf{PBS[P]}}(\lambda)\big]$$

is negligible in λ, where game UNF *is defined in Figure 2.*

In game UNF the adversary A gets a verification key vk as input and has access to two oracles $\textsc{Sign}_1$ and $\textsc{Sign}_2$. The oracles correspond to the two phases of the interactive protocol, representing an honest signer. The adversary can concurrently engage in polynomially many signing sessions for predicates of its choice to obtain blind signatures on messages. In order for A to win, it must output a non-empty vector $(m_i^*, \sigma_i^*)_{i \in [n]}$ of distinct and valid message/signature pairs; moreover, there must not exist an assignment of messages m_i^* to successfully closed sessions, that could "explain" the adversary's output. Formally, there must not exist an injective

```
$\mathsf{BLD}^{\mathsf{A},b}_{\mathsf{PBS[P]}}(\lambda)$
$par \leftarrow \mathsf{PBS.Setup}(1^\lambda)$
$(prd_0, prd_1, m_0, m_1, key, state) \leftarrow \mathsf{A}_1(par)$
if $\exists\, i, j \in \{0,1\}\colon \mathsf{P}(prd_i, m_j) = 0$ then
    return $0$
$(sess_0, sess_1) := (\mathtt{init}, \mathtt{init})$
$b' \leftarrow \mathsf{A}_2^{\textsc{User}_0,\textsc{User}_1,\textsc{User}_2}(state)$
return $b'$

$\textsc{User}_0(i)$
if $sess_i \neq \mathtt{init}$ then return $\bot$
$sess_i = \mathtt{open}$
$(msg, state_i)$
        $\leftarrow \mathsf{PBS.User}_0\big((par, key), prd_i, m_{i\oplus b}\big)$
return $msg$

$\textsc{User}_1(i, msg)$
if $sess_i \neq \mathtt{open}$ then return $\bot$
$sess_i = \mathtt{await}$
$(msg', state_i) \leftarrow \mathsf{PBS.User}_1(state_i, msg)$
return $msg'$

$\textsc{User}_2(i, msg)$
if $sess_i \neq \mathtt{await}$ then return $\bot$
$sess_i = \mathtt{closed}$
$\sigma_{i\oplus b} \leftarrow \mathsf{PBS.User}_2(state_i, msg)$
if $(sess_0 = sess_1 = \mathtt{closed})$ :
    if $(\sigma_0 = \bot \vee \sigma_1 = \bot)$ :
        return $(\bot, \bot)$
    return $(\sigma_0, \sigma_1)$
// in case other session is still open:
return $\varepsilon$
```

Fig. 3. The blindness game for a predicate blind signature scheme $\mathsf{PBS[P]}$ played by adversary $\mathsf{A} = (\mathsf{A}_1, \mathsf{A}_2)$. The operator "$\oplus$" is the XOR operation on bits, used to realize a swap of the message order if and only if $b = 1$.

mapping from the messages (m_i^*) to the predicates (prd_j) that were part of successfully closed signing sessions (stored by the experiment in the list $\vec{\textsc{Prd}}$), so that every message is mapped to a predicate it satisfies.[4]

Blindness. Blindness requires that whenever the signer gets to see one of its signatures, it cannot determine the session in which the signature was generated, except that it must have been in a session with a predicate that is satisfied by the message. We adapt the standard notion of blindness (see, e.g. [HKL19]) and let the adversary, in addition to two messages m_0 and m_1, define two predicates prd_0 and prd_1 (both satsified by m_0 and m_1). The challenger then chooses a bit b and runs the signing protocol with the adversary, asking for a signature on message m_b for predicate prd_0 and then for m_{1-b} for predicate prd_1. Being given the resulting signatures on m_0 and m_1, the adversary must determine the bit b.

Definition 13. *A predicate blind signature scheme* $\mathsf{PBS[P]}$ *satisfies* ***blindness*** *if for all adversaries* A

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

is negligible in λ*, where game* BLD *is defined in Figure 3.*

The game starts by letting the (adversarial) signer output two messages m_0 and m_1, predicates prd_0, prd_1, a verification key part key (i.e., what together with par gives vk), and a state $state$. The experiment then checks if both messages satisfy both predicates and returns 0 if this is not

[4] Note that checking if no such injective function exists is efficiently computable using e.g. the Hopcroft–Karp or Karzanov's matching algorithm [HK73, Kar73].

the case. Otherwise the experiment runs the signing protocol acting as the user (modeled via three different oracles $\textsc{User}_0, \textsc{User}_1, \textsc{User}_2$) to the signer's queries for two parallel sessions in total. The user oracles are set to ask for blind signatures on m_b with associated predicate prd_0 in the first session and m_{1-b} together with prd_1 in the second. If both signatures σ_1 and σ_2 obtained by $\mathsf{PBS.User}_2$ are not $\perp$ in the call of $\textsc{User}_2$, the signer is given (σ_0, σ_1) where σ_0 is a signature on m_0 and σ_1 a signature on m_1 under key vk. Blindness requires that after the completion of two sessions, there is no strategy for the signer that is noticeably better than just guessing the value of b.

We stress that all notions of blindness (full, partial or predicate blindness) only protect the user's privacy if at the time the user publishes his signature, the signer has blindly signed sufficiently many messages under the same key, and (in the case of partial blindness) same public message parts, and (in the case of predicate blindness) predicates satisfied by the message.

Hiding the predicates. By allowing the adversary A to output distinct predicates prd_0 and prd_1, we get an additional guarantee: namely, that the resulting message/signature pair does not reveal anything about the predicate that was used in the signing session (apart from the fact that the message satisfies it). This could be formalized via a game in which the adversary defines a message m and two predicates prd_0 and prd_1 (satisfied by m) and then plays the signer in a blind issuing of a signature on m using first prd_0 and then prd_1. If both sessions succeed, the adversary is given the signatures in random order, which the adversary has to determine. Showing that this notion is implied by blindness (Definition 13) is straightforward by a reduction that sets $m_0 := m$ and $m_1 := m$.

Predicate Blind Signatures Imply Partially Blind Signatures

Abe and Okamoto (AO) [AO00]) define *partially blind signatures* using the following syntax: messages consist of a *public part info* and a *secret part* m', and verification is of the form $\mathsf{Vfy}(vk, info, m, \sigma)$. When issuing a signature, signer and user agree on the public part.

Partially blind signatures can be constructed in a straightforward way from a predicate blind signature scheme for the following predicate family, which parses messages as pairs $(info, m')$, which we assume can be done unambiguously:

```
$\mathsf{P}(prd, m)$
$(info, m') := m$
**return** $info = prd$
```

To issue a signature for $info$ on a secret part m', the signer and user run the PBS signing protocol for $prd := info$ and the user sets the message to $m := (info, m')$. A signature σ for a pair $(info, m')$ is verified by running $\mathsf{Vfy}_{\mathsf{PBS}}(vk, (info, m), \sigma)$.

We show that unforgeability and blindness (as defined by AO [AO00]) of this construction follow from the respective notions for PBS defined in Definitions 12 and 13. To break unforgeability of a partially blind signature scheme, an adversary must output $\big(info, (m_i^*, \sigma_i^*)_{i\in[n]}\big)$, where the pairs (m_i^*, σ_i^*) are distinct and valid for public message part $info$ (i.e. $\mathsf{Vfy}(vk, info, m_i^*, \sigma_i^*) = 1$ for all $i \in [n]$), and the adversary queried the signing oracle $(n-1)$ times with public message part $info$.

Assuming an adversary A against this unforgeability notion for our construction, we define B for game UNF of the underlying PBS scheme that wins with equal probability. B runs A on the received key vk, and when A asks for a signature for public part *info*, B asks for a signature for predicate $prd := info$, relaying all of A's protocol messages *msg* and oracle replies msg'. When A returns $(info, (m'_i, \sigma_i)_{i\in[n]})$, the reduction B returns $\big(m^*_i := (info, m'_i), \sigma_i\big)_{i\in[n]}$.

If A wins then all (m'_i, σ_i) are distinct and valid w.r.t. *info*; therefore all $((info, m'_i), \sigma_i)$ are distinct and valid (under $\mathsf{Vfy}_{\mathsf{PBS}}$). Moreover, B has made at most $n-1$ queries for the predicate *info*, which is therefore contained in at most $n-1$ positions I in B's challenger's list $\vec{\text{PRD}}$. For all $j \notin I$, we have $\mathsf{P}(\vec{\text{PRD}}_j, m^*_i) = 0$ (since $\vec{\text{PRD}}_j \neq info$ and $m^*_i = (info, m'_i)$). Since $|I| \leq n-1$, there is no injective function ρ with $\mathsf{P}(\vec{\text{PRD}}_{\rho(i)}, m^*_i) = 1$ for all $i \in [n]$. Together, this means B has won UNF.

The definition of blindness by AO is similar to ours. One difference is that in AO the challenger samples a key pair (vk, sk) and gives it to the adversary, whereas in our definition the adversary can choose its own verification key part (which gives more realistic security guarantees). AO require the adversary to return two pairs $(info_0, m_0)$ and $(info_1, m_1)$ for which $info_0 = info_1$ holds. In our definition the adversary must output two messages and two predicates that are both satisfied by both messages. Since the predicates partition the message space, we must have $info_0 = info_1$. The reduction generates the key pair for the AO adversary and then simply forwards the oracle calls.

4 Predicate Blind Signatures with Schnorr Signatures

Signature issuing in "plain" blind Schnorr signatures (which do not satisfy Definition 12 [BLL+21]) works as follows. Let (x, X) be the signer's key pair. Similarly to computing a Schnorr signature, the signer first samples $r \leftarrow_\$ \mathbb{Z}_p$, computes $R := rG$ and sends it to the user. The user samples two blinding values $(\alpha, \beta) \leftarrow_\$ \mathbb{Z}_p^2$ and computes $R' := R + \alpha G + \beta X$, which will be the first component of the blind signature. The user then computes the corresponding value $c' := \mathsf{H}(R', X, m)$, blinds it to $c := (c' + \beta) \bmod p$ and sends c to the signer. The signer replies with $s := (r + cx) \bmod p$, which the user transforms to $s' := (s + \alpha) \bmod p$ and outputs the signature (R', s'). (See Eq. (1) below for why this yields a valid signature.)

To make this protocol secure, in our scheme the user must first, before receiving the value R, send an encryption C of the message m and the values α, β that the user will use. For this step we employ a public-key encryption scheme PKE. In her second message, together with c, the user also sends a zero-knowledge proof asserting that c was computed from these values m, α and β, and the signer will only send the final value s if this proof verifies. Since we also aim for a generalization to predicate blind signatures, the user's proof will also assert that the encrypted m satisfies the agreed-upon predicate *prd*. We therefore consider the following parameterized relation $\mathsf{R}_{\mathsf{PBS}}$:

$$\mathsf{R}_{\mathsf{PBS}}\Big(\overbrace{(p, \mathbb{G}, G, \mathsf{H})}^{par_{\mathsf{R}}}, \overbrace{(X, R, c, C, prd, ek)}^{\theta}, \overbrace{(m, \alpha, \beta, \rho)}^{w}\Big):$$

```
$R' := R + \alpha G + \beta X$                                // blind the signature group element $R$
return $\big(\mathsf{PKE.Enc}(ek, (m, \alpha, \beta); \rho) = C \ \wedge$    // $C$ encrypts witness elements under $ek$
        $c \equiv_p \mathsf{H}(R', X, m) + \beta \ \wedge$                  // $c$ is computed from witness elements
        $\mathsf{P}(prd, m) = 1\big)$                                 // $m$ satisfies the predicate $prd$
```

This relation $\mathsf{R}_{\mathsf{PBS}}$ checks, for given parameters $(p, \mathbb{G}, G, \mathsf{H})$, whether the user's message c was correctly computed for given X and R when the user's message is m and her randomness is α, β; whether the ciphertext C encrypts these values (m, α, β) using randomness ρ; and whether m satisfies the predicate *prd*. As the parameters of $\mathsf{R}_{\mathsf{PBS}}$ are exactly the Schnorr signature parameters, the relation-parameter sampling algorithm Rel for NArg is simply:

```
$\mathsf{NArg.Rel}(1^\lambda)$
$sp \leftarrow \mathsf{Sch.Setup}(1^\lambda)$
return $sp$
```

Let GrGen be a group generation algorithm and HGen be a hash function generator (which together define Sch.Setup as in Figure 1), let PKE be a public-key encryption scheme and P be a predicate compiler (which together define relation $\mathsf{R}_{\mathsf{PBS}}$), and let NArg be an argument system for $\mathsf{R}_{\mathsf{PBS}}$. Using the ideas sketched above, we obtain the 2-round predicate blind signature scheme $\mathsf{PBSch}[\mathsf{P}, \mathsf{GrGen}, \mathsf{HGen}, \mathsf{PKE}, \mathsf{NArg}]$ specified in Figure 4.

The message space $\mathcal{M}_{par}$ of PBSch can be arbitrary, as long as PKE can encrypt triples of the form (m, α, β). We therefore assume that for all λ, all $sp = (p, \mathbb{G}, G, \mathsf{H})$ output by $\mathsf{NArg.Rel}(1^\lambda)$, all *crs* output by $\mathsf{NArg.Setup}(sp)$ and all *ek* output by $\mathsf{PKE.KeyGen}(1^\lambda)$, we have $\mathcal{M}_{par} \times \mathbb{Z}_p \times \mathbb{Z}_p \subseteq \mathcal{M}_{ek}$ for $par := (crs, ek)$.

We now show that the construction in Figure 4 satisfies perfect correctness, unforgeability according to Definition 12 (proved in Appendix B) and blindness according to Definition 13 (proved in Appendix C).

Correctness. Perfect correctness follows from perfect correctness of the NArg and since a signature (R', s') obtained by the user after interacting with the signer satisfies PBSch.Ver (which is defined as Sch.Ver):

$$\begin{aligned} s'G = sG + \alpha G = (r + cx)G + \alpha G &= (r + \mathsf{H}(R', X, m)x + \beta x)G + \alpha G \\ &= R + \alpha G + \beta X + \mathsf{H}(R', X, m)X \\ &= R' + \mathsf{H}(R', X, m)\, X \ . \end{aligned} \tag{1}$$

Unforgeability. We bound the advantage in breaking the unforgeability of the predicate blind Schnorr signature construction by the advantages in breaking the security of the underlying primitives. Note that, in Assumption 1, we directly assume sEUF-CMA security of the Schnorr signature scheme. The reason is that all currently known security proofs of Schnorr signatures require the ROM [PS96, PS00, FPS20]. But the NArg relation $\mathsf{R}_{\mathsf{PBS}}$ depends on the used hash function, which would be replaced in the ROM by a random function. This situation occurs whenever a non-interactive argument is done over a relation that expresses operations of a primitive that can only be proven secure in the ROM. While Assumption 1 might be unconventional from a theoretical point of view, it is rather uncontroversial in a practical setting, given the wide-spread use of Schnorr signatures; and it is a *sine qua non* in any application involving Schnorr signatures anyway.

Theorem 1. *Let* P *be a predicate compiler and* GrGen *and* HGen *be a group and a hash generation algorithm; let* PKE *be a perfectly correct public-key encryption scheme and* $\mathsf{NArg}[\mathsf{R}_{\mathsf{PBS}}]$ *be a non-interactive argument scheme for the relation* $\mathsf{R}_{\mathsf{PBS}}$*. Then for any adversary* A *playing*

```
PBSch.Setup($1^\lambda$)
    $(p, \mathbb{G}, G) \leftarrow \mathsf{GrGen}(1^\lambda)$
    $\mathsf{H} \leftarrow \mathsf{HGen}(p)$
    $sp := (p, \mathbb{G}, G, \mathsf{H})$
    $(crs, \tau) \leftarrow \mathsf{NArg.Setup}(sp)$
    $(ek, dk) \leftarrow \mathsf{PKE.KeyGen}(1^\lambda)$
    $par := (crs, ek)$
    return $par$

PBSch.KeyGen($par$)
    $(p, \mathbb{G}, G) :\subseteq par$
    $x \leftarrow\!\!\$\, \mathbb{Z}_p\,;\ X := xG$
    $vk := (par, X)$
    $sk := (par, x)$
    return $(sk, vk)$

PBSch.Ver($vk, m, \sigma$)
    $(p, \mathbb{G}, G, \mathsf{H}, X) :\subseteq vk$
    $(R, s) := \sigma$
    $c := \mathsf{H}(R, X, m)$
    return $sG = R + cX$
```

```
PBSch.Sign($sk, prd$)                                  PBSch.User($vk, prd, m$)
$(p, \mathbb{G}, G, crs, ek, x) :\subseteq sk$          $(p, \mathbb{G}, G, \mathsf{H}, crs, ek, X) :\subseteq vk$
                                                       $\alpha, \beta \leftarrow\!\!\$\, \mathbb{Z}_p\,;\ \rho \leftarrow\!\!\$\, \mathcal{R}_{ek}$
                                                       $M := (m, \alpha, \beta)$
                              $\xleftarrow{C}$         $C := \mathsf{PKE.Enc}(ek, M; \rho)$
$r \leftarrow\!\!\$\, \mathbb{Z}_p\,;\ R := rG$        $\xrightarrow{R}$
                                                       $R' := R + \alpha G + \beta X$
                                                       $c := (\mathsf{H}(R', X, m) + \beta) \bmod p$
                                                       $\theta := (X, R, c, C, prd, ek)$
                                                       $w := (m, \alpha, \beta, \rho)$
                              $\xleftarrow{c, \pi}$    $\pi \leftarrow \mathsf{NArg.Prove}(crs, \theta, w)$
$\theta := (xG, R, c, C, prd, ek)$
if $\mathsf{NArg.Vfy}(crs, \theta, \pi) = 0$ :
    return 0
$s := (r + cx) \bmod p$        $\xrightarrow{s}$
                                                       if $sG \neq R + cX$ : return $\bot$
                                                       $s' := (s + \alpha) \bmod p$
return 1                                               return $\sigma := (R', s')$
```

Fig. 4. The *predicate blind Schnorr signature* scheme $\mathsf{PBSch}[\mathsf{P}, \mathsf{GrGen}, \mathsf{HGen}, \mathsf{PKE}, \mathsf{NArg}]$ based on a predicate compiler P, a group generation algorithm GrGen, a hash generator HGen, a public-key encryption scheme PKE and non-interactive zero-knowledge argument scheme $\mathsf{NArg}[\mathsf{R}_{\mathsf{PBS}}]$ for the relation $\mathsf{R}_{\mathsf{PBS}}$.

in game UNF *against the PBS scheme* $\mathsf{PBSch}[\mathsf{P}, \mathsf{GrGen}, \mathsf{HGen}, \mathsf{PKE}, \mathsf{NArg}]$ *defined in Figure 4, making at most q queries to the oracle* Sign_1*, there exist algorithms:*

- F *playing in game* sEUF-CMA *against the strong existential unforgeability of the Schnorr signature scheme* $\mathsf{Sch}[\mathsf{GrGen}, \mathsf{HGen}]$*,*
- S *playing in game* SND *against the soundness of* $\mathsf{NArg}[\mathsf{R}_{\mathsf{PBS}}]$*,*
- D *playing in game* DL *against the discrete-logarithm hardness of* GrGen*,*

s.t. for every $\lambda \in \mathbb{N}$:

$$\mathsf{Adv}^{\mathsf{UNF}}_{\mathsf{PBSch},\mathsf{A}}(\lambda) \leq \mathsf{Adv}^{\mathsf{sEUF\text{-}CMA}}_{\mathsf{Sch}[\mathsf{GrGen},\mathsf{HGen}],\mathsf{F}}(\lambda) + \mathsf{Adv}^{\mathsf{SND}}_{\mathsf{NArg}[\mathsf{R}_{\mathsf{PBS}}],\mathsf{S}}(\lambda) + q \cdot \mathsf{Adv}^{\mathsf{DL}}_{\mathsf{GrGen},\mathsf{D}}(\lambda) \ .$$

The proof of Theorem 1 can be found in Appendix B. The main idea is to reduce unforgeability of PBSch to unforgeability of Schnorr signatures. Given a verification key, the reduction sets up *crs* and the encryption key *ek* and simulates the signing oracles for the adversary. When the latter opens a signing session sending C, the reduction uses the decryption key corresponding to ek to decrypt C to m, α, and β. It then queries its own signing oracle for a signature $(\bar{R}, \bar{s})$ on m and sends $R \coloneqq \bar{R} - \alpha G - \beta X$ to the adversary. Upon receiving c, the accompanying proof π attests that it is consistent with m, α and β, which, by the definition of $\mathsf{R}_{\mathsf{PBS}}$, implies that $c \equiv_p \mathsf{H}(\bar{R}, X, m) + \beta$.

By the definition of Schnorr signing, letting $x \coloneqq \log X$ denote the secret key, we have $\bar{s} \equiv_p \log \bar{R} + \mathsf{H}(\bar{R}, X, m) \cdot x \equiv_p \log \bar{R} + c \cdot x - \beta \cdot x$. By the definition of PBSch, the adversary expects $s \equiv_p \log R + c \cdot x \equiv_p \log \bar{R} - \alpha - \beta \cdot x + c \cdot x$, which the reduction can compute as $s \coloneqq (\bar{s} - \alpha) \bmod p$.

Formally, the proof proceeds via a sequence of game hops. In the first hop, the experiment decrypts the user's ciphertext C and checks whether c is consistent with the plaintext (m, α, β) and that m satisfies the predicate. If not, the game aborts. We show that by perfect correctness of PKE, any abort can be used to break soundness of $\mathsf{NArg}[\mathsf{R}_{\mathsf{PBS}}]$. We then show that an adversary cannot compute a signature in a session which it has not closed, unless it breaks the discrete logarithm assumption (the factor q in the theorem statement comes from guessing in which of the signing sessions the adversary did so). Finally, we show that for any adversary that can still win, that is, there is no mapping of the adversary's messages to signing sessions, the adversary's output must contain a forgery for the Schnorr signature scheme.

Blindness. Perfect blindness of the "plain" blind Schnorr signature scheme is shown as follows [Sch01]: For every assignment of a resulting message/signature pair to a signature-issuing session, there exist unique values α and β that "explain" this assignment from the view of the signer. The following theorem shows that what our protocol adds to the "plain" variant does not reveal anything in a computational sense either.

Theorem 2. *Let* P *be a predicate compiler,* GrGen *and* HGen *be a group and hash generation algorithm; let* PKE *be a public-key encryption scheme and* $\mathsf{NArg}[\mathsf{R}_{\mathsf{PBS}}]$ *be an argument system for the relation* $\mathsf{R}_{\mathsf{PBS}}$*. Then for any adversary* A *playing in game* BLD *against the PBS scheme* $\mathsf{PBSch}[\mathsf{P}, \mathsf{GrGen}, \mathsf{HGen}, \mathsf{PKE}, \mathsf{NArg}]$ *defined in Figure 4, there exists algorithms:*

- Z_0 *and* Z_1*, playing in game* ZK *against* $\mathsf{NArg}[\mathsf{R}_{\mathsf{PBS}}]$*, and*
- C_0 *and* C_1*, playing in game* CPA *against* PKE*,*

s.t for every $\lambda \in \mathbb{N}$*:*

$$\mathsf{Adv}^{\mathsf{BLD}}_{\mathsf{PBSch},\mathsf{A}}(\lambda) \le \mathsf{Adv}^{\mathsf{ZK}}_{\mathsf{NArg}[\mathsf{R}_{\mathsf{PBS}}],\mathsf{Z}_0}(\lambda) + \mathsf{Adv}^{\mathsf{ZK}}_{\mathsf{NArg}[\mathsf{R}_{\mathsf{PBS}}],\mathsf{Z}_1}(\lambda) + \mathsf{Adv}^{\mathsf{CPA}}_{\mathsf{PKE},\mathsf{C}_0}(\lambda) + \mathsf{Adv}^{\mathsf{CPA}}_{\mathsf{PKE},\mathsf{C}_1}(\lambda) \ .$$

The proof of Theorem 2 can be found in Appendix C and proceeds via a sequence of game hops. Starting with game $\mathsf{BLD}^{\mathsf{A},b}_{\mathsf{PBS}[\mathsf{P}]}$ for an arbitrarily fixed b, we first replace the user's proofs π (in both signing sessions) by simulated proofs. We next replace the user's ciphertexts C by encryptions of a fixed message. These hops are indistinguishable by zero-knowledge of $\mathsf{NArg}[\mathsf{R}_{\mathsf{PBS}}]$ and CPA security of PKE. Now using the argument for plain blind Schnorr, this final game is independent of the bit b, which concludes the proof.

5 Design Choices and Implementation Details

In this section we discuss design choices of our predicate blind signature construction PBSch from Section 4, consider implementation details and discuss various trade-offs. Implementers can obtain different degrees of security and performance measures depending on the following:

- The protocol is intended to be used as an extension on top of an existing protocol that already implements Schnorr signatures for given parameters $(p, \mathbb{G}, G, \mathsf{H})$: This scenario applies to blockchain protocols that consider adding the predicate blind signature functionality. In such a case, migrating to new group parameters and/or a hash function is often impracticable or unachievable due to insufficient consensus.
- The Schnorr parameters $(p, \mathbb{G}, G, \mathsf{H})$ can be chosen in consideration of the specifics of the used scheme NArg: This scenario gives implementers far more flexibility regarding efficiency optimizations, particularly affecting the user's computational cost to perform $\mathsf{NArg.Prove}$, the size of the common reference string and the supported message length.

The first scenario faces two challenges. First, H is unlikely to be an "arithmetic-circuit-friendly" hash function [AGR+16, ACG+19, BGL20, AAB+20, GKR+21], meaning that expressing H in the language underlying $\mathsf{NArg}[\mathsf{R}_{\mathsf{PBS}}]$ will require a large number of addition and multiplication gates (and/or lookups). This in turn affects the efficiency of the blind signature issuing protocol. (This is for example the case for a standard choice for H like SHA256; see below.)

Second, the group $\mathbb{G}$, its elements and operations might not be efficiently representable in the circuit language of NArg (see Section 5.3 for a concrete example). The take-away point is that choosing the NArg first and then the Schnorr signature parameters $(p, \mathbb{G}, G, \mathsf{H})$ will potentially yield much more efficient instantiations.

5.1 Avoiding a Trusted Setup

In our security model for predicate blind signatures (Definitions 12 and 13) the parameters *par* are assumed to be generated in a trusted way, which in practice has to be dealt with. In our scheme PBSch (Figure 4), for a given security parameter λ and corresponding Schnorr parameters sp, $\mathsf{PBSch.Setup}$ generates a common reference string via $(crs, \tau) \leftarrow \mathsf{NArg.Setup}(sp)$ and a PKE encryption key via $(ek, dk) \leftarrow \mathsf{PKE.KeyGen}(1^\lambda)$.

The simulation trapdoor τ and the decryption key dk can be considered the protocol's "toxic waste" [COS20]. Any party that knows τ is able to simulate, and thus forge, proofs. Now if she engages in concurrent signing sessions, she can break unforgeability by mounting the attack [BLL+21] against the "plain" Schnorr blind-signing protocol [CP93]: during a signing session, the attacker commits to anything in the first round and then simulates the proof (of a false statement) in the second round. On the other hand, any party that knows dk is able to decrypt the user's ciphertexts and thereby completely break blindness.

Consequently, we can neither let the signer nor the user run $\mathsf{PBSch.Setup}$. If the signer runs it, this breaks blindness and thus the user's security; if the user runs it, then the signer's security, i.e., unforgeability, is at stake. The obvious solution might seem to let the signer run $\mathsf{NArg.Setup}$ and the user run $\mathsf{PKE.KeyGen}$. While the former is potentially insecure (see below), the latter is not practical, since typical application scenarios consider a single signer and multiple users.

PKE SETUP. The problem of every user generating their own encryption key can be overcome by generating a single *ek transparently*, that is, in a way so no corresponding secret key is known to any party. When the public key *ek* is a group element and the secret key its discrete logarithm $dk = \log_G(ek)$, as e.g. for ElGamal encryption [ElG85], this can be established easily by "hashing into the group" [BF01, BCI+10]. An agreed-upon public string is hashed to obtain the public key.

NIZK setup. As with the PKE part, we can also simply instantiate NArg with a scheme with a transparent setup, of which there now exists a host, such as *Ligero* [AHIV17], *Bulletproofs* [BBB⁺18], *Hyrax* [WTs⁺18], *Aurora* [BCR⁺19], *STARKs* [BBHR19], *DARK* [BFS19], *Halo* [BGH19], *RedShift* [KPV19], *Spartan* [Set20], *Fractal* [COS20], *Xiphos* and *Kopis* [SL20].

A particularly efficient SNARK scheme, with the shortest proofs of all schemes, is *Groth16* [Gro16], which however requires a trusted setup, since it has a "structured" common reference string (CRS). If we let the signer set up her own CRS, then she is protected against attacks against soundness. However, for blindness, the user relies on NArg being zero-knowledge, a property that also assumes that the CRS was set up in a trusted way.

The solution to this dilemma is to use a *subversion zero-knowledge* proof scheme [BFS16], which protects the user against malicious signers. The notion guarantees that even when the CRS is maliciously set up, the prover is guaranteed that a proof computed w.r.t. it will not leak anything about the used witness (and thus blindness still holds). For *Groth16* Fuchsbauer [Fuc18] defines an algorithm to check that a CRS is well-formed and he shows that if provers only accept well-formed CRSs, subversion zero knowledge holds. On the other hand, a malformed CRS enables attacks against *Groth16*, where proofs constructed using such a CRS leak information on the witness [CGGN17, Fuc19]. Once a CRS is checked, the scheme can be used as usual.

5.2 Practical Considerations for the Relation

In this section we address theoretical and practical aspects concerning the parameterized relation $\mathsf{R}_{\mathsf{PBS}}(par_{\mathsf{R}}, \theta, w) \to 0/1$ defined in Section 4, and start with stressing the importance of type checks on elements in the witness w, as their insufficiency or absence provides a common source of error in implementations. Type checks on elements of the statement θ can either be expressed inside the relation, which leads to increased prover time and CRS size, or alternatively, they can be omitted in the relation but then have to be performed by the verifier alongside to NIZK proof verification, where they cause proof rejection if any check fails.

Complexity of the Supported Predicates. Since most NIZK schemes (and in particular efficient zk-SNARKs) in the literature are realized as proofs of circuit satisfiability for a circuit that expresses the relation (where the parameters par_{R} are hardwired and the statement θ as well as the witness w are inputs to the circuit), the concrete complexity of the predicate compiler P in the language native to NArg directly affects proving time and common reference string size (while proof size and verifier time is unaffected for certain choices of NArg).[5]

In practice, low complexity for P might be achieved by keeping the number of supported predicates small,[6] e.g. by considering a set $\{\mathsf{Prd}_i\}_{i\in[n]}$, and define the compiler P as:

```
$\mathsf{P}(prd, m)$
----------------------------------
**if** $prd = 1$ : **return** $\mathsf{Prd}_1(m)$
                 $\vdots$
**if** $prd = n$ : **return** $\mathsf{Prd}_n(m)$
**return** $0$
```

[5] One could for example consider predicates prd that attest that m (or parts of it) constitutes an accepting (zk)SNARK proof and thereby obtain protocols with high degrees of expressibility.

[6] For scenarios where only a single predicate prd is supported by P, this predicate can obviously be hardwired.

Maximum generality can be achieved by letting P be the *universal circuit* that interprets the input *prd* as a circuit description (a.k.a. a circuit-SAT instance) and emulates its execution on its second input m. In practice this would result in an inefficient scheme, since it requires to encode the universal circuit in terms of the language native to the underlying NIZK.

Hardwiring for Efficiency. Efficiency can be improved by moving elements of the statement θ into the parameters par_{R}. This is because circuit complexity generally reduces when values are circuit constants rather than inputs (e.g. multiplication by constants vs. multiplication by variables). On the other hand, with increased hardwiring, flexibility and reusability of the CRS reduces. Implementers can thus choose to instantiate NArg for a relation out of a spectrum of relations lying somewhere between minimal hardwiring Eq. (2), and maximal hardwiring Eq. (3).

Minimal Hardwiring, and thus maximal flexibility, is the setting in which we defined our scheme in Section 4:

$$\mathsf{R}_{\mathsf{PBS}}\big(\overbrace{(p, \mathbb{G}, G, \mathsf{H})}^{par_{\mathsf{R}}}, \overbrace{(X, R, c, C, prd, ek)}^{\theta}, \overbrace{(m, \alpha, \beta, \rho)}^{\omega}\big) \ . \tag{2}$$

An practical advantage is that the same CRS, and thus scheme parameters *par*, can be used by multiple signers since they are independent of the signature verification keys X.

This has another benefit when using a subversion-zero-knowledge scheme, such as *Groth16*, to improve efficiency, which however requires a (potentially complex) CRS-consistency check [Fuc19]. The more a CRS is used, the faster it is expected to be recognized and reported if it is malformed; users can thus have confidence in a CRS even when they don't check consistency themselves. The downside is that, since the signer's security relies on the secrecy of the simulation trapdoor, the CRS would have to be set up in a "ceremony" using multiparty computation [BGM17] – or one uses one of the tranparent schemes mentioned in Section 5.1.

A theoretical advantage of minimal hardwiring is that unforgeability of the PBS can be reduced to standard soundness of NArg. In Theorem 1, the reduction against soundness receives the CRS and creates the signature and PKE key pairs (x, X) and (ek, dk) itself. It thus knows the values x and dk required to simulate the game.

Maximal Hardwiring corresponds to a relation of the form:

$$\mathsf{R}'_{\mathsf{PBS}}\big(\overbrace{(p, \mathbb{G}, G, \mathsf{H}, X, ek)}^{par_{\mathsf{R}'}}, \overbrace{(R, c, C, prd)}^{\theta}, \overbrace{(m, \alpha, \beta, \rho)}^{\omega}\big) \ , \tag{3}$$

where we move the signature verification key X and the encryption key ek to the relation parameters $par_{\mathsf{R}'}$. An immediate consequence is improved verification time, since NArg verifier time grows at least linearly in the statement size. It moreover reduces circuit complexity and therefore prover time and CRS size. This is the recommended setting when signers generate their own CRS and thus need not worry about proper deletion of the simulation trapdoor.

From a theoretical point of view, including X and ek in $par_{\mathsf{R}'}$ requires allowing *auxiliary input* in the definition of soundness of argument systems (Definition 4). This means that the relation generator $\mathsf{NArg.Rel}$ can output auxiliary information *aux* in addition to the relation parameters, which the soundness adversary gets as input in addition to *crs*. This notion of soundness is standard but stronger than Definition 4, since one needs to argue that the auxiliary input comes from a distribution that does not undermine soundness [BCPR14].

Concretely, the relation generator for $\mathsf{R}'_{\mathsf{PBS}}$ in Eq. (3) runs $(p, \mathbb{G}, G, \mathsf{H}) \leftarrow \mathsf{Sch.Setup}(1^\lambda)$ (as for $\mathsf{R}_{\mathsf{PBS}}$), and in addition $(ek, dk) \leftarrow \mathsf{PKE.KeyGen}(1^\lambda)$; $x \leftarrow_\$ \mathbb{Z}_p$ and $X \coloneqq xG$. It returns (x, dk) as auxiliary input. In the proof of unforgeability (Theorem 1), when reducing to soundness of NArg, the reduction thus still has the values x and dk it requires to simulate the game.

5.3 Blind Schnorr Using secp256k1 and SHA-256

We briefly consider the scenario where $\mathbb{G}$ is the standardized elliptic curve secp256k1 group and H is the hash function SHA-256, as is the case for Bitcoin since the *Taproot* upgrade by Blockstream [WNR20]. The curve secp256k1 is defined by the equation $\mathcal{E} : y^2 = x^3 + 7$ over the finite field of size $p = 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1$. The group order $q = |\mathcal{E}(\mathbb{F}_p)|$ is prime and the curve's embedding degree does not permit an efficient pairing, which excludes pairing-based SNARK schemes [Gro16, GWC19, SL20] as potential candidates, unless one can afford to pay the high cost in performance of conducting non-native modulo-p reductions in the field generated by the pairing-friendly curve [KPS18], in order to compute the group element $R' \coloneqq R + \alpha G + \beta X$ and check if $c \equiv_p \mathsf{H}(R', X, m) + \beta$.

As observed in [SS11], the sec**p**256k1 curve has a "twin" defined over the same equation, which is called sec**q**256k1 and whose order is p. This "cycle of curves" enables the use of a proof system that does not require pairings and can be instantiated over the sec**q**256k1 curve, such as *Bulletproofs* [BBB$^+$18] or *Halo* [BGH19]. This avoids expensive overhead in expressing modulo-p reductions in the circuit and hence group operations of the curve sec**p**256k1 are relatively cheap to arithmetize. The factors of the group order

$$\begin{aligned} |\mathbb{F}_p^*| = p - 1 = 2 \cdot 3 \cdot 7 \cdot 13441 \cdot \\ \mathrm{0x1db8260e5e3b460a46a0088fccf6a3a5936d75d89a776d4c0da4f338aafb} \end{aligned} \tag{4}$$

however imply that there is no support for run-time-optimal FFT techniques, but require a method such as Schönhage and Strassen's algorithm [SS71], which runs in $O(n \log n \log \log n)$, instead of $O(n \log n)$ for n gates[7], but is still feasible for practical purposes. Transparent-setup zkSNARKs such as *Redshift* [KPV19], *STARKs* [BBHR19] or *Aurora* [BCR$^+$19], that depend on the IOPP protocol FRI [BBHR18] when committing to the circuit assignment and forgo the use of elliptic curves, face a similar problem. That is, to avoid expressing the modulo-p reduction explicitly, the FRI protocol has to be instantiated over $\mathbb{F}_p$. But since the multiplicative subgroup $\mathbb{F}_p^*$ requires high 2-adicity for FRI's folding technique, this is not possible considering the factors in Eq. (4). To the best of our knowledge, it remains an open problem if FRI (which shares many features of the FFT algorithm) can be efficiently executed over non-smooth domains.

As mentioned earlier, SHA-256 is a prominent example of a function that is not known to be *arithmetic-circuit-friendly*. As a result, one call of its compression function requires around 26 000 R1CS constraints [CGGN17, KPS18], which approximately amounts to 110 000 gates in the 2-fan-in arithmetization used by *Halo* and *Bulletproofs*.

We currently recommend to implement our predicate blind Schnorr signature scheme using *Halo 2* [Com21], the extension of *Halo*. *Halo 2* also has a transparent setup and can be instantiated over the curve cycle sec**p**256k1–sec**q**256k1. It supports batch verification, recursive

[7] There exist more refined techniques for such scenarios such as [HvdH22, BCKL21], which achieve better asymptotics but whose concrete constants are likely to render them unpractical.

proof composition via proof accumulation and due to its "*PLONK*-ish arithmetization"[GWC19] it supports lookups, which become particularly useful when multiple invocations of binary operations (as it is the case for SHA-256) are required.

A disadvantage of *Halo 2* compared to *Bulletproofs* comes from its reliance on the permutation argument of *PLONK* [GWC19], which requires a multiplicative subgroup of order $\geq n$. If n is smaller than the group order, the circuit needs to be extended by dummy gates to match the group order. Considering the factors of $|\mathbb{F}_p^*|$ in Eq. (4), we obtain an upper bound for supported circuit sizes of $2 \cdot 3 \cdot 7 \cdot 13441 = 564\,522$ gates, which is sufficient to execute a few invocations of the compression function while leaving enough space to consider predicate deciders P of decent complexity. We remark that the support for efficient proof accumulation (and therefore recursion) could be exploited to overcome the gate-number limitation in certain aspects and allow for example arbitrary-length messages by implementing the Merkle-Damgård construction that underlies the SHA-256 in a recursive circuit design over the curve cycle sec**p**256k1–sec**q**256k1.

Acknowledgements. The authors are supported by the Vienna Science and Technology Fund (WWTF) through project VRG18-002. We would like to thank Tim Ruffing for preliminary discussions.

References

[AAB$^+$20] Abdelrahaman Aly, Tomer Ashur, Eli Ben-Sasson, Siemen Dhooghe, and Alan Szepieniec. Design of symmetric-key primitives for advanced cryptographic protocols. *IACR Trans. Symm. Cryptol.*, 2020(3):1–45, 2020.

[ACG$^+$19] Martin R. Albrecht, Carlos Cid, Lorenzo Grassi, Dmitry Khovratovich, Reinhard Lüftenegger, Christian Rechberger, and Markus Schofnegger. Algebraic cryptanalysis of STARK-friendly designs: Application to MARVELlous and MiMC. In Steven D. Galbraith and Shiho Moriai, editors, *ASIACRYPT 2019, Part III*, volume 11923 of *LNCS*, pages 371–397. Springer, December 2019.

[AGR$^+$16] Martin R. Albrecht, Lorenzo Grassi, Christian Rechberger, Arnab Roy, and Tyge Tiessen. MiMC: Efficient encryption and cryptographic hashing with minimal multiplicative complexity. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part I*, volume 10031 of *LNCS*, pages 191–219. Springer, December 2016.

[AHIV17] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 2087–2104. ACM Press, October / November 2017.

[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, August 2000.

[App] Apple. iCloud private relay. Available at https://www.apple.com/privacy/docs/iCloud_Private_Relay_Overview_Dec2021.PDF.

[BBB$^+$18] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy*, pages 315–334. IEEE Computer Society Press, May 2018.

[BBHR18] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Fast reed-solomon interactive oracle proofs of proximity. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *ICALP 2018*, volume 107 of *LIPIcs*, pages 14:1–14:17. Schloss Dagstuhl, July 2018.

[BBHR19] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable zero knowledge with no trusted setup. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 701–732. Springer, August 2019.

[BCC88] Gilles Brassard, David Chaum, and Claude Crépeau. Minimum disclosure proofs of knowledge. *J. Comput. Syst. Sci.*, 37(2):156–189, 1988.

[BCC+09] Mira Belenkiy, Jan Camenisch, Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Hovav Shacham. Randomizable proofs and delegatable anonymous credentials. In Shai Halevi, editor, *CRYPTO 2009*, volume 5677 of *LNCS*, pages 108–125. Springer, August 2009.

[BCI+10] Eric Brier, Jean-Sébastien Coron, Thomas Icart, David Madore, Hugues Randriam, and Mehdi Tibouchi. Efficient indifferentiable hashing into ordinary elliptic curves. In Tal Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 237–254. Springer, August 2010.

[BCKL09] Mira Belenkiy, Melissa Chase, Markulf Kohlweiss, and Anna Lysyanskaya. Compact e-cash and simulatable VRFs revisited. In Hovav Shacham and Brent Waters, editors, *PAIRING 2009*, volume 5671 of *LNCS*, pages 114–131. Springer, August 2009.

[BCKL21] Eli Ben-Sasson, Dan Carmon, Swastik Kopparty, and David Levit. Elliptic curve Fast Fourier Transform (ECFFT) part I: Fast polynomial algorithms over all finite fields. *Electron. Colloquium Comput. Complex.*, 28:103, 2021.

[BCPR14] Nir Bitansky, Ran Canetti, Omer Paneth, and Alon Rosen. On the existence of extractable one-way functions. In David B. Shmoys, editor, *46th ACM STOC*, pages 505–514. ACM Press, May / June 2014.

[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, May 2019.

[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.

[BDN18] Dan Boneh, Manu Drijvers, and Gregory Neven. Compact multi-signatures for smaller blockchains. In Thomas Peyrin and Steven Galbraith, editors, *ASIACRYPT 2018, Part II*, volume 11273 of *LNCS*, pages 435–464. Springer, December 2018.

[Ber15] Daniel J. Bernstein. Multi-user schnorr security, revisited. Cryptology ePrint Archive, Paper 2015/996, 2015. https://eprint.iacr.org/2015/996.

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

[BFL20] Balthazar Bauer, Georg Fuchsbauer, and Julian Loss. A classification of computational assumptions in the algebraic group model. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part II*, volume 12171 of *LNCS*, pages 121–151. Springer, August 2020.

[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.

[BFP21] Balthazar Bauer, Georg Fuchsbauer, and Antoine Plouviez. The one-more discrete logarithm assumption in the generic group model. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part IV*, volume 13093 of *LNCS*, pages 587–617. Springer, December 2021.

[BFS16] Mihir Bellare, Georg Fuchsbauer, and Alessandra Scafuro. NIZKs with an untrusted CRS: Security in the face of parameter subversion. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part II*, volume 10032 of *LNCS*, pages 777–804. Springer, December 2016.

[BFS19] Benedikt Bünz, Ben Fisch, and Alan Szepieniec. Transparent snarks from dark compilers. *IACR Cryptol. ePrint Arch.*, 2019:1229, 2019.

[BGH19] Sean Bowe, Jack Grigg, and Daira Hopwood. Halo: Recursive proof composition without a trusted setup. Cryptology ePrint Archive, Report 2019/1021, 2019. https://eprint.iacr.org/2019/1021.

[BGL20] Eli Ben-Sasson, Lior Goldberg, and David Levit. STARK friendly hash – survey and recommendation. Cryptology ePrint Archive, Report 2020/948, 2020. https://eprint.iacr.org/2020/948.

[BGM17] Sean Bowe, Ariel Gabizon, and Ian Miers. Scalable multi-party computation for zk-SNARK parameters in the random beacon model. Cryptology ePrint Archive, Report 2017/1050, 2017. https://eprint.iacr.org/2017/1050.

[BK22] Dan Boneh and Chelsea Komlo. Threshold signatures with private accountability. In *CRYPTO 2022, Part IV*, LNCS, pages 551–581. Springer, August 2022.

[BL13] Foteini Baldimtsi and Anna Lysyanskaya. Anonymous credentials light. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 2013*, pages 1087–1098. ACM Press, November 2013.

[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, October 2021.

[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.

[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, January 2003.

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby, editors, *ACM CCS 93*, pages 62–73. ACM Press, November 1993.

[Bra94] Stefan Brands. Untraceable off-line cash in wallets with observers (extended abstract). In Douglas R. Stinson, editor, *CRYPTO'93*, volume 773 of *LNCS*, pages 302–318. Springer, August 1994.

[BRS20] Samuel Brack, Leonie Reichert, and Björn Scheuermann. CAUDHT: Decentralized contact tracing using a DHT and blind signatures. Cryptology ePrint Archive, Report 2020/398, 2020. https://eprint.iacr.org/2020/398.

[CFN90] David Chaum, Amos Fiat, and Moni Naor. Untraceable electronic cash. In Shafi Goldwasser, editor, *CRYPTO'88*, volume 403 of *LNCS*, pages 319–327. Springer, August 1990.

[CGGN17] Matteo Campanelli, Rosario Gennaro, Steven Goldfeder, and Luca Nizzardo. Zero-knowledge contingent payments revisited: Attacks and payments for services. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 229–243. ACM Press, October / November 2017.

[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.

[Cha88] David Chaum. Elections with unconditionally-secret ballots and disruption equivalent to breaking RSA. In C. G. Günther, editor, *EUROCRYPT'88*, volume 330 of *LNCS*, pages 177–182. Springer, May 1988.

[CHM$^+$20] Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Noah Vesely, and Nicholas P. Ward. Marlin: Preprocessing zkSNARKs with universal and updatable SRS. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part I*, volume 12105 of *LNCS*, pages 738–768. Springer, May 2020.

[CL01] Jan Camenisch and Anna Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In Birgit Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 93–118. Springer, May 2001.

[Com21] The Electric Coin Company. The halo2 book, 2021. Available at https://zcash.github.io/halo2/index.html.

[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, May 2020.

[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, August 1993.

[DJW22] Frank Denis, Frederic Jacobs, and Christopher A. Wood. RSA blind signatures [work in progress], 2022. Available at https://datatracker.ietf.org/doc/draft-irtf-cfrg-rsa-blind-signatures/.

[DLZ$^+$20] Aaqib Bashir Dar, Auqib Hamid Lone, Saniya Zahoor, Afshan Amin Khan, and Roohie Naaz. Applicability of mobile contact tracing in fighting pandemic (COVID-19): Issues, challenges and solutions. Cryptology ePrint Archive, Report 2020/484, 2020. https://eprint.iacr.org/2020/484.

[ElG85] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.

[FHS15] Georg Fuchsbauer, Christian Hanser, and Daniel Slamanig. Practical round-optimal blind signatures in the standard model. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 233–253. Springer, August 2015.

[FKL18] Georg Fuchsbauer, Eike Kiltz, and Julian Loss. The algebraic group model and its applications. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 33–62. Springer, August 2018.

[FKP17] Manuel Fersch, Eike Kiltz, and Bertram Poettering. On the one-per-message unforgeability of (EC)DSA and its variants. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part II*, volume 10678 of *LNCS*, pages 519–534. Springer, November 2017.

[FO22] Georg Fuchsbauer and Michele Orrù. Non-interactive Mimblewimble transactions, revisited. To appear at ASIACRYPT'22, 2022. Available at https://eprint.iacr.org/2022/265.

[FOO93] Atsushi Fujioka, Tatsuaki Okamoto, and Kazuo Ohta. A practical secret voting scheme for large scale elections. In Jennifer Seberry and Yuliang Zheng, editors, *AUSCRYPT'92*, volume 718 of *LNCS*, pages 244–251. Springer, December 1993.

[FOS19] Georg Fuchsbauer, Michele Orrù, and Yannick Seurin. Aggregate cash systems: A cryptographic investigation of Mimblewimble. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part I*, volume 11476 of *LNCS*, pages 657–689. Springer, May 2019.

[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, May 2020.

[Fuc11] Georg Fuchsbauer. Commuting signatures and verifiable encryption. In Kenneth G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 224–245. Springer, May 2011.

[Fuc18] Georg Fuchsbauer. Subversion-zero-knowledge SNARKs. In Michel Abdalla and Ricardo Dahab, editors, *PKC 2018, Part I*, volume 10769 of *LNCS*, pages 315–347. Springer, March 2018.

[Fuc19] Georg Fuchsbauer. WI is not enough: Zero-knowledge contingent (service) payments revisited. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 49–62. ACM Press, November 2019.

[GG14] Sanjam Garg and Divya Gupta. Efficient round optimal blind signatures. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 477–495. Springer, May 2014.

[GKM+18] Jens Groth, Markulf Kohlweiss, Mary Maller, Sarah Meiklejohn, and Ian Miers. Updatable and universal common reference strings with applications to zk-SNARKs. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part III*, volume 10993 of *LNCS*, pages 698–728. Springer, August 2018.

[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.

[GO94] Oded Goldreich and Yair Oren. Definitions and properties of zero-knowledge proof systems. *Journal of Cryptology*, 7(1):1–32, December 1994.

[Goo] Google. VPN by Google One. Available at https://one.google.com/about/vpn/howitworks.

[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, May 2016.

[GWC19] A. Gabizon, Zachary J. Williamson, and Oana Ciobotaru. Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. *IACR Cryptol. ePrint Arch.*, 2019:953, 2019.

[HAB+17] Ethan Heilman, Leen Alshenibr, Foteini Baldimtsi, Alessandra Scafuro, and Sharon Goldberg. TumbleBit: An untrusted bitcoin-compatible anonymous payment hub. In *NDSS 2017*. The Internet Society, February / March 2017.

[Her97] Mark Allan Herschberg. *Secure electronic voting over the world wide web*. PhD thesis, Massachusetts Institute of Technology, 1997.

[HK73] John E Hopcroft and Richard M Karp. An nˆ5/2 algorithm for maximum matchings in bipartite graphs. *SIAM Journal on computing*, 2(4):225–231, 1973.

[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, May 2019.

[HKOK06] Yoshikazu Hanatani, Yuichi Komano, Kazuo Ohta, and Noboru Kunihiro. Provably secure electronic cash based on blind multisignature schemes. In Giovanni Di Crescenzo and Avi Rubin, editors, *FC 2006*, volume 4107 of *LNCS*, pages 236–250. Springer, February / March 2006.

[HvdH22] David Harvey and Joris van der Hoeven. Polynomial multiplication over finite fields in time $o(n \log n$. *Journal of the ACM (JACM)*, 69:1 – 40, 2022.

[JLO97] Ari Juels, Michael Luby, and Rafail Ostrovsky. Security of blind digital signatures (extended abstract). In Burton S. Kaliski Jr., editor, *CRYPTO'97*, volume 1294 of *LNCS*, pages 150–164. Springer, August 1997.

[Kar73] Alexander V Karzanov. An exact estimate of an algorithm for finding a maximum flow, applied to the problem on representatives. *Problems in Cybernetics*, 5:66–70, 1973.

[KLX22] Julia Kastner, Julian Loss, and Jiayu Xu. On pairing-free blind signature schemes in the algebraic group model. In Goichiro Hanaoka, Junji Shikata, and Yohei Watanabe, editors, *PKC 2022, Part II*, volume 13178 of *LNCS*, pages 468–497. Springer, 2022.

[KPS18] Ahmed E. Kosba, Charalampos Papamanthou, and Elaine Shi. xJsnark: A framework for efficient verifiable computation. In *2018 IEEE Symposium on Security and Privacy*, pages 944–961. IEEE Computer Society Press, May 2018.

[KPV19] Assimakis Kattis, Konstantin Panarin, and Alexander Vlasov. RedShift: Transparent SNARKs from list polynomial commitment IOPs. Cryptology ePrint Archive, Report 2019/1400, 2019. https://eprint.iacr.org/2019/1400.

[LLL$^+$19] Yi Liu, Zhen Liu, Yu Long, Zhiqiang Liu, Dawu Gu, Fei Huan, and Yanxue Jia. TumbleBit++: A comprehensive privacy protocol providing anonymity and amount-invisibility. In Ron Steinfeld and Tsz Hon Yuen, editors, *ProvSec 2019*, volume 11821 of *LNCS*, pages 339–346. Springer, October 2019.

[MPSW19] Gregory Maxwell, Andrew Poelstra, Yannick Seurin, and Pieter Wuille. Simple Schnorr multi-signatures with applications to Bitcoin. *Des. Codes Cryptogr.*, 87(9):2139–2164, 2019.

[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, November 2016.

[Nec94] V. I. Nechaev. Complexity of a determinate algorithm for the discrete logarithm. *Mathematical Notes*, 55(2):165–172, 1994.

[Nic19] Jonas Nick. Blind signatures in scriptless scripts. Presentation given at *Building on Bitcoin* 2019, 2019. Slides and video available at https://jonasnick.github.io/blog/2018/07/31/blind-signatures-in-scriptless-scripts/.

[NS02] Phong Q. Nguyen and Igor Shparlinski. The insecurity of the digital signature algorithm with partially known nonces. *Journal of Cryptology*, 15(3):151–176, June 2002.

[Oka06] Tatsuaki Okamoto. Efficient blind and partially blind signatures without random oracles. In Shai Halevi and Tal Rabin, editors, *TCC 2006*, volume 3876 of *LNCS*, pages 80–99. Springer, March 2006.

[OO92] Tatsuaki Okamoto and Kazuo Ohta. Universal electronic cash. In Joan Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 324–337. Springer, August 1992.

[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, May 1996.

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

[ROG07] Francisco Rodríguez-Henríquez, Daniel Ortiz-Arroyo, and Claudia García-Zamora. Yet another improvement over the Mu–Varadharajan e-voting protocol. *Computer Standards & Interfaces*, 29(4):471–480, 2007.

[Sch90] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 239–252. Springer, August 1990.

[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, November 2001.

[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, August 2020.

[Sho97] Victor Shoup. Lower bounds for discrete logarithms and related problems. In Walter Fumy, editor, *EUROCRYPT'97*, volume 1233 of *LNCS*, pages 256–266. Springer, May 1997.

[SL20] Srinath Setty and Jonathan Lee. Quarks: Quadruple-efficient transparent zkSNARKs. Cryptology ePrint Archive, Report 2020/1275, 2020. https://eprint.iacr.org/2020/1275.

[SS71] Arnold Schönhage and Volker Strassen. Schnelle Multiplikation grosser Zahlen. *Computing*, 7(3):281–292, 1971.

[SS11] Joseph H Silverman and Katherine E Stange. Amicable pairs and aliquot cycles for elliptic curves. *Experimental Mathematics*, 20(3):329–357, 2011.

[Wag02] David Wagner. A generalized birthday problem. In Moti Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, pages 288–303. Springer, August 2002.

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

[WTs$^+$18] Riad S. Wahby, Ioanna Tzialla, abhi shelat, Justin Thaler, and Michael Walfish. Doubly-efficient zkSNARKs without trusted setup. In *2018 IEEE Symposium on Security and Privacy*, pages 926–943. IEEE Computer Society Press, May 2018.

A Weak OMDL

We introduce the weak one-more discrete logarithm (wOMDL) problem as a stepping stone in our proof of unforgeability of our predicate blind signature construction in Appendix B. The wOMDL problem consists in computing the discrete logarithm of q group elements, obtained from a challenge oracle, while being given access to a discrete-logarithm oracle that can be called at most $q - 1$ times. In contrast to the original OMDL game [BNPS03], the DL oracle can *only be queried on challenge group elements*, and not on arbitrary group elements. This makes the wOMDL assumption significantly weaker than OMDL. We show that wOMDL is implied by DL, while such an implication is unlikely to hold for OMDL [BFL20].

Definition 14. *A group generation algorithm* GrGen *satisfies the* ***weak one-more-discrete logarithm assumption*** *if for every p.p.t. adversary* A

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

is negligible in λ*, where the game* wOMDL *is defined by:*

```
$\mathsf{wOMDL}^{\mathsf{A}}_{\mathsf{GrGen}}(1^\lambda)$
  $(p, \mathbb{G}, G) \leftarrow \mathsf{GrGen}(1^\lambda)$
  $\vec{x} := [\,]$; $k := 0$
  $\vec{y} \leftarrow \mathsf{A}^{\textsc{Chal},\textsc{DLog}}(p, \mathbb{G}, G)$
  return $\big(\vec{y} = \vec{x} \ \wedge\ k < |\vec{x}|\big)$

$\textsc{Chal}()$
  $x \leftarrow\$ \mathbb{Z}_p$ ; $X := xG$
  $\vec{x} = \vec{x} \| x$
  return $X$

$\textsc{DLog}(i)$
  $k := k + 1$
  return $\vec{x}_i$
```

Lemma 1. *For every p.p.t. algorithm* A *playing in game* wOMDL *that calls the* $\textsc{Chal}$ *oracle at most q times, there exists a p.p.t. algorithm* B *playing in game* DL *s.t.*

$$\mathsf{Adv}^{\mathsf{wOMDL}}_{\mathsf{GrGen},\mathsf{A}}(\lambda) = q \cdot \mathsf{Adv}^{\mathsf{DL}}_{\mathsf{GrGen},\mathsf{B}}(\lambda)\ . \tag{5}$$

Proof. We construct B playing against DL, which on input $(p, \mathbb{G}, G, Z)$ must compute $\log_G(Z)$. B simulates game wOMDL for A, except that for a random, say the j-the $\textsc{Chal}()$ query, B embeds Z and aborts if A queries $\textsc{DLog}(j)$. If A wins wOMDL outputting $\vec{y}$, its j-th component is the DL solution.

```
$\mathsf{B}(p, \mathbb{G}, G, Z)$
  $\vec{x} := [\,]$ // empty list
  $j \leftarrow\$ [q]$
  $\vec{y} \leftarrow \mathsf{A}^{\textsc{Chal},\textsc{DLog}}(p, \mathbb{G}, G)$
  return $\vec{y}_j$

$\textsc{Chal}()$
  if $|\vec{x}| = j - 1$ : // i.e., $j$-th $\textsc{Chal}()$ call
    $\vec{x} = \vec{x} \| \bot$
    return $Z$
  $x \leftarrow\$ \mathbb{Z}_p$ ; $X := xG$
  $\vec{x} = \vec{x} \| x$
  return $X$

$\textsc{DLog}(i)$
  if $\vec{x}_i = \bot$ :
    abort
  return $\vec{x}_i$
```

As there must be at least one index $i \in [|\vec{x}|]$ so that A has not queried $\textsc{DLog}(i)$, and since $|\vec{x}| \le q$, the probability of B not aborting is at least $1/q$, which proves the lemma. □

B Proof of Theorem 1

We give a formal proof that our predicate blind signature scheme PBSch from Figure 4 satisfies strong unforgeability according to Definition 12 by providing reductions to the security properties of its building blocks. We proceed by a sequence of games specified in Figure 5.

```
$\mathsf{UNF}^{\mathsf{A}}_{\mathsf{PBSch}}(\lambda)$, $\boxed{\mathsf{G}_1}$, $\boxed{\mathsf{G}_2}$, $\boxed{\mathsf{G}_3}$

$(p, \mathbb{G}, G, \mathsf{H}) \leftarrow \mathsf{Sch.Setup}(1^\lambda)$
$(crs, \tau) \leftarrow \mathsf{NArg.Setup}(p, \mathbb{G}, G, \mathsf{H})$
$(ek, dk) \leftarrow \mathsf{PKE.KeyGen}(1^\lambda)$
$x \leftarrow\$ \mathbb{Z}_p$ ; $X \coloneqq xG$
$vk \coloneqq (crs, ek, X)$
$\vec{\mathrm{S}} \coloneqq [\,]$; $\vec{\textsc{Prd}} \coloneqq [\,]$
$\vec{\mathrm{M}} \coloneqq [\,]; \vec{\alpha} \coloneqq [\,]; \vec{\beta} \coloneqq [\,];$
$\vec{\mathrm{D}} \coloneqq [\,]$; $\mathcal{Q} \coloneqq \emptyset$
$\big(m_i^*, \sigma_i^* \coloneqq (R_i^*, s_i^*)\big)_{i \in [n]} \leftarrow \mathsf{A}^{\textsc{Sign}_1, \textsc{Sign}_2}(vk)$
if $\big(\exists i \in [n]\ \exists j \in [|\vec{\mathrm{S}}|] : m_i^* = \vec{\mathrm{M}}_j \wedge$
        $\vec{\mathrm{S}}_j \neq \varepsilon \wedge \vec{\mathrm{S}}_j[0] + \vec{\alpha}_j G + \vec{\beta}_j X = R_i^*\big)$ :
    return 0                                                        (II)
return $\big(\ n > 0$
    $\wedge\, \forall i \in [n]$: $\mathsf{PBSch.Ver}(vk, m_i^*, \sigma_i^*) = 1$
    $\wedge\, \forall i \neq j \in [n]$: $(m_i^*, \sigma_i^*) \neq (m_j^*, \sigma_j^*)$
    $\wedge\, \nexists \rho \in \mathsf{InjF}([n], [|\vec{\textsc{Prd}}|])$:
            $\forall i \in [n]$: $\mathsf{P}(\vec{\textsc{Prd}}_{\rho(i)}, m_i^*) = 1\big)$
```

```
$\textsc{Sign}_1(prd, C)$

$r \leftarrow\$ \mathbb{Z}_p$ ; $R \coloneqq rG$
$(m, \alpha, \beta) \coloneqq \mathsf{PKE.Dec}(dk, C)$
$\vec{\mathrm{M}} = \vec{\mathrm{M}} \| m$ ; $\vec{\alpha} = \vec{\alpha} \| \alpha$ ; $\vec{\beta} = \vec{\beta} \| \beta$
$(\bar{R}, \bar{s}) \leftarrow \mathsf{Sch.Sign}\big((p, \mathbb{G}, G, \mathsf{H}, x), m\big)$
$\mathcal{Q} = \mathcal{Q} \,\|\, \big(m, (\bar{R}, \bar{s})\big)$
$R = \bar{R} - \alpha G - \beta X$
$\vec{\mathrm{D}} = \vec{\mathrm{D}} \| \big((\bar{s} - \alpha) \bmod p\big)$
$\vec{\mathrm{S}} = \vec{\mathrm{S}} \| (R, r, C, prd)$
return $R$
```

```
$\textsc{Sign}_2\big(j, (c, \pi)\big)$

if $\vec{\mathrm{S}}_j = \varepsilon$ : return $\bot$
$(R, r, C, prd) \coloneqq \vec{\mathrm{S}}_j$
$\theta \coloneqq (X, R, c, C, prd, ek)$
if $\mathsf{NArg.Vfy}(crs, \theta, \pi) = 0$ :
    return 0
$R' \coloneqq R + \vec{\alpha}_j G + \vec{\beta}_j X$
if $\big(c \not\equiv_p (\mathsf{H}(R', X, \vec{\mathrm{M}}_j) + \vec{\beta}_j) \vee \mathsf{P}(prd, \vec{\mathrm{M}}_j) \neq 1\big)$ :
    abort and return 0                                              (I)
$\vec{\mathrm{S}}_j \coloneqq \varepsilon$; $\vec{\textsc{Prd}} = \vec{\textsc{Prd}} \| prd$
return $\vec{\mathrm{D}}_j$
return $\big((r + cx) \bmod p\big)$
```

Fig. 5. The unforgeability game from Figure 2 for the scheme $\mathsf{PBSch}[\mathsf{P}, \mathsf{GrGen}, \mathsf{HGen}, \mathsf{PKE}, \mathsf{NArg}]$ from Figure 4 and hybrid games used in the proof of Theorem 1. G_i includes all boxes with an index $\leq i$ and ignores all boxes with and index $> i$.

$\underline{\mathsf{G}_0}$ is game UNF from Figure 2 with PBS instantiated by PBSch from Figure 4. The generic $\mathsf{PBS.Setup}$ hence is replaced by the setup from Figure 4. In $\textsc{Sign}_1$, the call $\mathsf{PBS.Sign}_1$ is instantiated by sampling $r \leftarrow\$ \mathbb{Z}_p$ and returning $R = rG$, and in $\textsc{Sign}_2$, we instantiate $\mathsf{PBS.Sign}_2$ as defined in Figure 4, by a NIZK verification and return 0 if verification failed.

$\underline{\mathsf{G}_1}$. In G_1 we introduce three lists $\vec{\mathrm{M}}$, $\vec{\alpha}$ and $\vec{\beta}$ and modify $\textsc{Sign}_1$, so that on each call with input (prd, C) we decrypt C to obtain the values (m, α, β), which we then append to the lists $\vec{\mathrm{M}} = \vec{\mathrm{M}} \| m$, $\vec{\alpha} = \vec{\alpha} \| \alpha$, $\vec{\beta} = \vec{\beta} \| \beta$. In each $\textsc{Sign}_2$ call on input $(j, (c, \pi))$, we check if for the decrypted values at index j, we either have $c \not\equiv_p \mathsf{H}(R', X, \vec{\mathrm{M}}_j) + \vec{\beta}_j$ for $R' \coloneqq R + \vec{\alpha}_j G + \vec{\beta}_j X$, or $\mathsf{P}(prd, \vec{\mathrm{M}}_j) = 0$. If either is the case, we stop the game and return 0.

REDUCTION TO SOUNDNESS OF NArg. We show that the difference in $\mathsf{Adv}^{\mathsf{UNF}}_{\mathsf{PBSch},\mathsf{A}}(\lambda)$ and $\mathsf{Adv}^{\mathsf{G}_1}_{\mathsf{A}}(\lambda)$ is bounded by the advantage in winning the game SND (Definition 4) against soundness of $\mathsf{NArg}[\mathsf{R}_{\mathsf{PBS}}]$ played by adversary S defined in Figure 6.

```
$\mathsf{S}(crs)$
    $(p, \mathbb{G}, G, \mathsf{H}) :\subseteq crs$
    $(ek, dk) \leftarrow \mathsf{PKE.KeyGen}(1^\lambda)$
    $x \leftarrow\!\$\, \mathbb{Z}_p$ ;   $X := xG$
    $vk := (crs, ek, X)$
    $\vec{\mathrm{S}} := [\,]$; $\vec{\mathrm{M}} := [\,]; \vec{\alpha} := [\,]; \vec{\beta} := [\,]$
    $(m_i^*, \sigma_i^*)_{i\in[n]} \leftarrow \mathsf{A}^{\textsc{Sign}_1,\textsc{Sign}_2}(vk)$
    return $\perp$

$\textsc{Sign}_1(prd, C)$
    $r \leftarrow\!\$\, \mathbb{Z}_p$ ;  $R := rG$
    $(m, \alpha, \beta) := \mathsf{PKE.Dec}(dk, C)$
    $\vec{\mathrm{M}} = \vec{\mathrm{M}} \| m$ ;  $\vec{\alpha} = \vec{\alpha} \| \alpha$ ;  $\vec{\beta} = \vec{\beta} \| \beta$
    $\vec{\mathrm{S}} = \vec{\mathrm{S}} \| (R, r, C, prd)$
    return $R$

$\textsc{Sign}_2\big(j, (c, \pi)\big)$
    if $\vec{\mathrm{S}}_j = \varepsilon$ : return $\perp$
    $(R, r, C, prd) := \vec{\mathrm{S}}_j$
    $\theta := (X, R, c, C, prd, ek)$
    if $\mathsf{NArg.Vfy}(par, \theta, \pi) = 0$ :
        return 0
    $R' := R + \vec{\alpha}_j G + \vec{\beta}_j X$
    if $\big(c \not\equiv_p (\mathsf{H}(R', X, \vec{\mathrm{M}}_j) + \vec{\beta}_j) \vee \mathsf{P}(prd, \vec{\mathrm{M}}_j) \neq 1\big)$ :
        stop and return $(\theta, \pi)$
    $\vec{\mathrm{S}}_j := \varepsilon$
    return $\big((r + cx) \bmod p\big)$
```

Fig. 6. Adversary S playing against soundness of $\mathsf{NArg}[\mathsf{R}_{\mathsf{PBS}}]$

According to the definition of game SND for NArg for relation $\mathsf{R}_{\mathsf{PBS}}$, S gets as input the common reference string crs, generated by $\mathsf{NArg.Setup}(par_{\mathsf{R}})$, where par_{R} is generated by $\mathsf{NArg.Rel}$, which is defined as $\mathsf{Sch.Setup}$. Reduction S, run in game SND therefore perfectly simulates G_1 to adversary A until abort. In $\textsc{Sign}_2$, it checks whether for a valid statement/proof pair (θ, π) parts of the supposed witness m, α and β satisfy $c \neq (\mathsf{H}(R', X, m) + \beta) \bmod p$ or $\mathsf{P}(prd, m) \neq 1$ and returns the pair (θ, π), if either is the case. S thus returns a pair (θ, π) if and only if G_1 aborts in line (I). It remains to show that when this happens, S wins game SND. Assume S reaches the "stop" condition in a call $\textsc{Sign}_2$ on input $\big(j, (c, \pi)\big)$ and let $\big(\theta := (X, R, c, C, prd, ek), \pi\big)$ be its output. The condition is only reached if π is an accepting proof for statement θ. It suffices thus to show that θ is not a valid statement.

Towards contradiction, assume θ is a valid statement, meaning that there exists $w' := (m', \alpha', \beta', \rho')$ s.t. $\mathsf{R}_{\mathsf{PBS}}(par_{\mathsf{R}}, \theta, w') = 1$, which means:

$$\mathsf{PKE.Enc}(ek, (m', \alpha', \beta'); \rho') = C \;\wedge\; c \equiv_p \mathsf{H}(R', X, m') + \beta' \;\wedge\; \mathsf{P}(prd, m') = 1\ , \tag{6}$$

where $R' := R + \alpha' G + \beta' X$. By definition of S we have $(m, \alpha, \beta) = \mathsf{PKE.Dec}(dk, C)$. By perfect correctness of PKE, together with the first clause in Eq. (6), this can only be the case if

$$m' = m \quad \text{and} \quad \alpha' = \alpha \quad \text{and} \quad \beta' = \beta\ . \tag{7}$$

Again by the definition of S, we have $c \not\equiv_p \mathsf{H}(R + \alpha G + \beta X, X, m) + \beta \ \vee\ \mathsf{P}(prd, m) \neq 1$, which is a contradiction to Eq. (6) and (7). Therefore such a witness w' does not exist. This means that whenever (I) is reached in G_1, then S wins SND and thus

$$\mathsf{Adv}^{\mathsf{UNF}}_{\mathsf{PBSch},\mathsf{A}}(\lambda) \leq \mathsf{Adv}^{\mathsf{SND}}_{\mathsf{NArg}[\mathsf{R}_{\mathsf{PBS}}],\mathsf{S}}(\lambda) + \Pr\left[\mathsf{G}_1^{\mathsf{A}}(\lambda)\right] . \tag{8}$$

<u>G_2.</u> In G_2 we introduce the event

$$\mathsf{E} \ :\Leftrightarrow\ \exists\, i \in [n]\ \exists\, j \in [|\vec{\mathrm{S}}|] : m_i^* = \vec{\mathrm{M}}_j \ \wedge\ \vec{\mathrm{S}}_j \neq \varepsilon \ \wedge\ \vec{\mathrm{S}}_j[0] + \vec{\alpha}_j G + \vec{\beta}_j X = R_i^* \ , \tag{9}$$

which we check after the adversary made its final output, and return 0 if it is satisfied. If E occurs, our final reduction to the unforgeability of Schnorr signatures will not work, since A might only return signatures that the reduction asked to its signing oracle. Concretely, the event E states that for at least one of the messages m_i^* in A's final output, there exists a session j where this particular message was decrypted in $\textsc{Sign}_1$ (formalized by $\vec{\mathrm{M}}_j = m_i^*$) and session j was not successfully closed via a call to $\textsc{Sign}_2$ (formalized by $\vec{\mathrm{S}}_j \neq \varepsilon$) and yet the first part of the message's Schnorr signature R_i^* is related to $R_j := \vec{\mathrm{S}}_j[0]$ that was returned in the j-th $\textsc{Sign}_1$ call s.t. $R_j + \vec{\alpha}_j G + \vec{\beta}_j X = R_i^*$, where $\vec{\alpha}_j$ and $\vec{\beta}_j$ were obtained via decryption in the j-th session. We have $\Pr\left[\mathsf{G}_2^{\mathsf{A}}(\lambda)\right] = \Pr\left[\mathsf{G}_1^{\mathsf{A}}(\lambda) \wedge \neg\mathsf{E}\right]$. Together with $\Pr\left[\mathsf{G}_1^{\mathsf{A}}(\lambda)\right] = \Pr\left[\mathsf{G}_1^{\mathsf{A}}(\lambda) \wedge \mathsf{E}\right] + \Pr\left[\mathsf{G}_1^{\mathsf{A}}(\lambda) \wedge \neg\mathsf{E}\right]$ we obtain:

$$\Pr\left[\mathsf{G}_1^{\mathsf{A}}(\lambda)\right] = \Pr\left[\mathsf{G}_1^{\mathsf{A}}(\lambda) \wedge \mathsf{E}\right] + \Pr\left[\mathsf{G}_2^{\mathsf{A}}(\lambda)\right] . \tag{10}$$

<u>Reduction to DL.</u> We bound $\Pr\left[\mathsf{G}_1^{\mathsf{A}}(\lambda) \wedge \mathsf{E}\right]$ 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 Figure 7; then we apply Lemma 1 to reduce to the hardness of DL.

By the definition of game wOMDL, L receives as input the group parameters $(p, \mathbb{G}, G)$. With that it chooses a hash function $\mathsf{H} \leftarrow \mathsf{HGen}(p)$ and then simulates G_1 for A, where in each call of $\textsc{Sign}_1$, L queries its challenge oracle $R \leftarrow \textsc{Chal}()$. 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 $\textsc{Sign}_2$, L obtains $r \leftarrow \textsc{DLog}(j)$ from its oracle $\textsc{DLog}$.

To analyze the difference in success probabilities of G_1 and G_2, we assume L satisfies condition E from (9) and outputs $\vec{r}$. If E occurs, we have that some session number j with challenge $R_j := \vec{\mathrm{S}}_j[0]$ was not closed, and thus the oracle $r_j \leftarrow \textsc{DLog}(j)$ was not called either. Also for some index $i \in [n]$, we have $R_j + \vec{\alpha}_j G + \vec{\beta}_j X = R_i^*$ and by the assertion that A wins G_1, we know by the validity of the signatures that $s_i^* G = R_i^* + \mathsf{H}(R_i^*, X, m_i^*) X$. Combining these equations yields $s_i^* G = r_j G + \vec{\alpha}_j G + \vec{\beta}_j x G + \mathsf{H}(R_i^*, X, m_i^*) x G$, and thus $s_i^* \equiv_p r_j + \vec{\alpha}_j + \vec{\beta}_j x + \mathsf{H}(R_i^*, X, m_i^*) x$. From this, L computes $r_j := \log_G(R_j)$ and for all other challenges R_t, $t \neq j$, for which the session has not been closed, L calls $\textsc{DLog}(t)$ and stores the reply r_j in the list $\vec{r}$. By outputting $\vec{r}$, L wins game wOMDL, since $\vec{r}$ contains the discrete logarithm of all challenges and the oracle $\textsc{DLog}$ has been called $|\vec{r}| - 1$ times as required by the game.

Therefore we obtain:

$$\Pr\left[\mathsf{G}_1^{\mathsf{A}}(\lambda) \wedge \mathsf{E}\right] \leq \mathsf{Adv}^{\mathsf{wOMDL}}_{\mathsf{GrGen},\mathsf{L}} . \tag{11}$$

```
$\mathsf{L}^{\textsc{Chal},\textsc{DLog}}(p, \mathbb{G}, G)$
$\mathsf{H} \leftarrow \mathsf{HGen}(p)$
$(crs, \tau) \leftarrow \mathsf{NArg.Setup}\big((p, \mathbb{G}, G, \mathsf{H})\big)$
$(ek, dk) \leftarrow \mathsf{PKE.KeyGen}(1^\lambda)$
$x \leftarrow\!\!\$\, \mathbb{Z}_p \;;\; X := xG$
$vk := (crs, ek, X)$
$\vec{\mathrm{S}} := [\,];\ \vec{\textsc{Prd}} := [\,]$
$\vec{\mathrm{M}} := [\,]; \vec{\alpha} := [\,]; \vec{\beta} := [\,]$
$\vec{r} := [\,]$
$\big(m_i^*, (R_i^*, s_i^*)\big)_{i \in [n]} \leftarrow \mathsf{A}^{\textsc{Sign}_1, \textsc{Sign}_2}(vk)$
if $\big(\exists i \in [n]\ \exists j \in [|\vec{\mathrm{S}}|] : m_i^* = \vec{\mathrm{M}}_j \wedge$
    $\vec{\mathrm{S}}_j \neq \varepsilon \wedge \vec{\mathrm{S}}_j[0] + \vec{\alpha}_j G + \vec{\beta}_j X = R_i^*\big)$ :
    for $t \in [|\vec{\mathrm{S}}|] \setminus \{j\}$ :
        if $\vec{\mathrm{S}}_t \neq \varepsilon$ :
            // close all open sessions except $j$-th
            $\vec{r}_t \leftarrow \textsc{DLog}(j)$
    $\vec{r}_j := \big(s_i^* - \vec{\alpha}_j - \vec{\beta}_j \cdot x$
                $- \mathsf{H}(R_i^*, X, m_i^*) \cdot x\big) \bmod p$
    return $\vec{r}$
return $\perp$
```

```
$\textsc{Sign}_1(prd, C)$
$R \leftarrow \textsc{Chal}()$
$(m, \alpha, \beta) := \mathsf{PKE.Dec}(dk, C)$
$\vec{\mathrm{M}} = \vec{\mathrm{M}} \| m \,;\ \vec{\alpha} = \vec{\alpha} \| \alpha \,;\ \vec{\beta} = \vec{\beta} \| \beta$
$\vec{\mathrm{S}} = \vec{\mathrm{S}} \| (R, C, prd)$
$\vec{r} = \vec{r} \| 0$    // keep $|\vec{r}|$ consistent with other lists
$R' := R + \alpha G + \beta X$
return $R$
```

```
$\textsc{Sign}_2\big(j, (c, \pi)\big)$
if $\vec{\mathrm{S}}_j = \varepsilon$ : return $\perp$
$(R, C, prd) := \vec{\mathrm{S}}_j$
$\theta := (X, R, c, C, prd, ek)$
if $\mathsf{NArg.Vfy}(crs, \theta, \pi) = 0$ :
    return $\perp$
$R' := R + \vec{\alpha}_j G + \vec{\beta}_j X$
if $\big(c \not\equiv_p (\mathsf{H}(R', X, \vec{\mathrm{M}}_j) + \vec{\beta}_j) \vee \mathsf{P}(prd, \vec{\mathrm{M}}_j) \neq 1\big)$ :
    abort and return 0                                (I)
$\vec{\mathrm{S}}_j := \varepsilon; \vec{\textsc{Prd}} = \vec{\textsc{Prd}} \| prd$
$\vec{r}_j := \textsc{DLog}(j)$
return $\big((\vec{r}_j + cx) \bmod p\big)$
```

Fig. 7. Adversary L playing in game wOMDL from Definition 14

Now let q be the upper-bound of queries to the $\textsc{Sign}_1$ oracle made by A. Then L's number of queries to its $\textsc{Chal}$ oracle is also bounded by q, and by applying Lemma 1 we obtain an adversary D playing in the game DL where

$$\Pr\left[\mathsf{G}_1^{\mathsf{A}}(\lambda) \wedge \mathsf{E}\right] \leq q \cdot \mathsf{Adv}^{\mathsf{DL}}_{\mathsf{GrGen},\mathsf{D}} \,. \tag{12}$$

$\underline{\mathsf{G}_3}$. In G_3 we prepare the reduction to sEUF-CMA security of the Schnorr signature scheme $\mathsf{Sch}[\mathsf{GrGen}, \mathsf{HGen}]$ underlying PBSch, by making the following changes: First we introduce two empty lists $\vec{\mathrm{D}}$ and $\mathcal{Q}$.

Then we modify $\textsc{Sign}_1$ so that after decrypting C to (m, α, β), we compute a Schnorr signature on m under the signing key $sk := (p, \mathbb{G}, G, \mathsf{H}, x)$ by running $(\bar{R}, \bar{s}) \leftarrow \mathsf{Sch.Sign}(sk, m)$. Next we replace the random signer challenge $R := rG$ by $R := \bar{R} - \alpha G - \beta X$. Note that $\mathsf{Sch.Sign}$ returns a uniform $\bar{R}$ and hence R is a uniform element and thus the simulation is perfect up to here. As a last change in $\textsc{Sign}_1$, we compute and append $(\bar{s} - \alpha) \bmod p$ to the list $\vec{\mathrm{D}}$. In $\textsc{Sign}_2$ instead of returning $s := (r + cx) \bmod p$ we return the value we previously stored in $\vec{\mathrm{D}}$, that is $s := \bar{s} - \alpha = \vec{\mathrm{D}}_j$.

The user now obtains simulated elements $(R, s) = (\bar{R} - \alpha G - \beta X, \bar{s} - \alpha)$. By definition of $\mathsf{Sch.Sign}$ we have $\bar{s}G = \bar{R} + \mathsf{H}(\bar{R}, X, m)X$, and by the assertion that line (I) was not reached, we have $c \equiv_p \mathsf{H}(R + \alpha G + \beta X, X, m) + \beta$. We show that for any choice of α, β and message m and signing key sk, the user's new view is distributed equivalently to the old view. According to the definition of G_2, the latter is

$$\begin{aligned}&\{(R, s) \mid r \leftarrow\!\!\$\, \mathbb{Z}_p\,;\; R = rG\,;\; s \equiv_p r + (\mathsf{H}(R + \alpha G + \beta X, X, m) + \beta)x\}\\ &\qquad\qquad \equiv \{(\bar{R} - \alpha G - \beta X, s) \mid \bar{r} \leftarrow\!\!\$\, \mathbb{Z}_p\,;\; \bar{R} = \bar{r}G\,;\; s \equiv_p \bar{r} - \alpha - \beta x + \mathsf{H}(\bar{R}, X, m)x + \beta x\}\end{aligned}$$

(since $\bar{r} - \alpha - \beta x$ is distributed as r; setting $\bar{s} = s + \alpha$, this is distributed as follows)

$$\begin{aligned}&\equiv \{(\bar{R} - \alpha G - \beta X, \bar{s} - \alpha) \mid \bar{r} \leftarrow\!\!\$\, \mathbb{Z}_p\,;\; \bar{R} = \bar{r}G\,;\; \bar{s} \equiv_p \bar{r} + \mathsf{H}(\bar{R}, X, m)x\}\\ &\equiv \{(\bar{R} - \alpha G - \beta X, \bar{s} - \alpha) \mid (\bar{R}, \bar{s}) \leftarrow \mathsf{Sch.Sign}(sk, m)\}\ ,\end{aligned}$$

which is precisely the view in G_3. Thus the simulation remains perfect and we obtain:

$$\Pr\left[\mathsf{G}_2^{\mathsf{A}}(\lambda)\right] = \Pr\left[\mathsf{G}_3^{\mathsf{A}}(\lambda)\right]\ . \tag{13}$$

<u>REDUCTION TO sEUF-CMA SECURITY OF STANDARD SCHNORR SIGNATURE.</u> To finish the proof, we construct adversary F in Figure 8 that succeeds in the game sEUF-CMA against the Schnorr signature scheme $\mathsf{Sch}[\mathsf{GrGen}, \mathsf{HGen}]$ with probability $\Pr\left[\mathsf{G}_3^{\mathsf{A}}(\lambda)\right]$.

```
$\mathsf{F}^{\textsc{Sign}}\big((sp, X)\big)$
    $(crs, \tau) \leftarrow \mathsf{NArg.Setup}(sp)$
    $(ek, dk) \leftarrow \mathsf{PKE.KeyGen}(1^\lambda)$
    $vk := (crs, ek, X)$
    $\vec{\mathrm{S}} := [\,]\,;\; \vec{\textsc{Prd}} := [\,]$
    $\vec{\mathrm{M}} := [\,]; \vec{\alpha} := [\,]; \vec{\beta} := [\,];$
    $\vec{\mathrm{D}} := [\,]; \mathcal{Q} := [\,]$
    $\mathcal{F} \leftarrow \mathsf{A}^{\textsc{Sign}_1, \textsc{Sign}_2}(vk)$
    $(m^*, \sigma^*) \leftarrow\!\!\$\, \mathcal{F} \setminus \mathcal{Q}$
    return $(m^*, \sigma^*)$

$\textsc{Sign}_1(prd, C)$
    $(m, \alpha, \beta) := \mathsf{PKE.Dec}(dk, C)$
    $\vec{\mathrm{M}} = \vec{\mathrm{M}} \| m\,;\; \vec{\alpha} = \vec{\alpha} \| \alpha\,;\; \vec{\beta} = \vec{\beta} \| \beta$
    $(\bar{R}, \bar{s}) \leftarrow \textsc{Sign}(m)$
    $\mathcal{Q} = \mathcal{Q} \,\|\, \big(m, (\bar{R}, \bar{s})\big)$
    $R := \bar{R} - \alpha G - \beta X$
    $\vec{\mathrm{D}} = \vec{\mathrm{D}} \| \big((\bar{s} - \alpha) \bmod p\big)$
    $\vec{\mathrm{S}} = \vec{\mathrm{S}} \| (R, r, C, prd)$
    return $R$
```

Fig. 8. F paying against sEUF-CMA security of $\mathsf{Sch}[\mathsf{GrGen}, \mathsf{HGen}]$. The oracle $\textsc{Sign}_2$ is simulated to A is as defined in game G_3 in Figure 2.

By the definition of sEUF-CMA, F receives as challenge input a Schnorr verification key $(sp, X) := vk$ and has access to a signing oracle $\textsc{Sign}$. With the Schnorr parameters sp it completes $\mathsf{PBSch.Setup}$ computing the common reference string crs for NArg and a key pair (ek, dk) for PKE. Moreover, F initializes a list $\mathcal{Q}$ used to store the message/signature pairs from its signing oracle $\textsc{Sign}$. When F simulates G_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 F on each $\textsc{Sign}_1$ query by A forwards the call to its signing oracle $\textsc{Sign}$. Since the challenge key X in the sEUF-CMA game is a uniform random element, the simulation remains perfect.

We show that if A wins G_3 outputting $\mathcal{F} = (m_i^*, \sigma_i^*)_{i\in[n]}$, then this set must contain a successful forgery for F, that is, an element that is not contained in $\mathcal{Q} = (m_j, \sigma_j \coloneqq (\bar{R}_j, \bar{s}_j))_{j\in[|\vec{\mathrm{S}}|]}$ (where index j corresponds to the signing session number in which the pair was added to $\mathcal{Q}$). Letting J be the set of indices of the sessions that were eventually closed, we can define $\mathcal{Q}_{\text{cls}} \coloneqq (m_j, \sigma_j)_{j\in J}$.

We first show that there exists an element $(m_{i^*}^*, \sigma_{i^*}^*) \in \mathcal{F}$ that is not in $\mathcal{Q}_{\text{cls}}$. If we had $\mathcal{F} \subseteq \mathcal{Q}_{\text{cls}}$ then there would exist and injective function $\rho\colon [n] \to J$ mapping elements of $\mathcal{F}$ to elements of $\mathcal{Q}_{\text{cls}}$, in particular, $m_i^* = m_{\rho(i)}$. For all $j \in J$ (the closed sessions), we have $\vec{\mathrm{PRD}}_j(m_j) = 1$, as otherwise G_3 would have aborted in line (I). We thus have $1 = \vec{\mathrm{PRD}}_{\rho(i)}(m_{\rho(i)}) = \vec{\mathrm{PRD}}_{\rho(i)}(m_i^*)$ for all $i \in [n]$, which contradicts the winning condition of G_3, which requires that no such ρ exists.

We next show that $(m_{i^*}^*, \sigma_{i^*}^*) \notin \mathcal{Q} \setminus \mathcal{Q}_{\text{cls}}$, that is, it was not obtained in an unfinished session either. Towards a contradiction, assume for some $j \notin J$: $(m_{i^*}^*, (R_{i^*}^*, s_{i^*}^*)) = (m_j, (\bar{R}_j, \bar{s}_j))$. Then we would have (a) $m_{i^*}^* = m_j = \vec{\mathrm{M}}_j$ (since $\vec{\mathrm{M}}$ stores the same messages as $\mathcal{Q}$), (b) $\vec{\mathrm{S}}_j \neq \bot$ (since the session was not closed), and (considering the value R in the definition of $\textsc{Sign}_1$) $\vec{\mathrm{S}}_j[0] = \bar{R}_j - \vec{\alpha}_j G - \vec{\beta}_j X$, which together with $R_{i^*}^* = \bar{R}_j$ yields (c) $R_{i^*}^* = \vec{\mathrm{S}}_j[0] + \vec{\alpha}_j G + \vec{\beta}_j X$. Now the existence of values i^* and j with (a)–(c) leads precisely to an abort of G_3 in line (II).

We have thus shown that $(m_{i^*}^*, \sigma_{i^*}^*)$ is neither in $\mathcal{Q}_{\text{cls}}$, nor in $\mathcal{Q} \setminus \mathcal{Q}_{\text{cls}}$, and thus not in $\mathcal{Q}$, which means it is thus a valid forgery for F. We have thus:

$$\Pr\big[\mathsf{G}_3^{\mathsf{A}}(\lambda)\big] \leq \mathsf{Adv}^{\mathsf{sEUF\text{-}CMA}}_{\mathsf{Sch}[\mathsf{GrGen},\mathsf{HGen}],\mathsf{F}}(\lambda)\ . \tag{14}$$

Putting Eq. (8)–(14) together we obtain Theorem 1. □

C Proof of Theorem 2

We give a formal proof that our predicate blind signature scheme PBSch from Figure 4 satisfies blindness as defined in Definition 13. The proofs works via reductions to the security of the underlying building blocks, that is, the zero-knowledge property of NArg and CPA-security of the scheme PKE. For succinctness and readability we sometimes omit the security parameter λ in the proof but keep it as an implicit input to the games and advantage definitions. We proceed by a sequence of games specified in Figure 9.

$\underline{\mathsf{G}_0}$ is game BLD from Figure 3 with PBS instantiated with PBSch from Figure 4, that is, $\mathsf{PBS.Setup}$, $\mathsf{PBS.User}_0$, $\mathsf{PBS.User}_1$ and $\mathsf{PBS.User}_2$ are replaced by the instantiations defined in Figure 4. The variables $state_0$ and $state_1$ in BLD are replaced by the session variables $\alpha_i, \beta_i, \rho_i, R'_i, c_i, C_i$ for both sessions $i \in \{0, 1\}$. As $\alpha_i, \beta_i, \rho_i$ are uniform values, we can sample them right away. $R'_{i\oplus b}$ is part of $state_i$, but we renamed it since in $\textsc{User}_2$ it becomes part of $\sigma_{i\oplus b}$.

$\underline{\mathsf{G}_1}$. In G_1 we make the following change: On oracle call $\textsc{User}_1$, instead of creating a proof via $\mathsf{NArg.Prove}$ we use the simulator $\mathsf{NArg.SimProve}$ to simulate a proof for the statement θ. We show that this change is not efficiently noticeable by defining adversaries Z_0 and Z_1 in Figure 10 that play in game ZK against the $\mathsf{NArg}[\mathsf{R}_{\mathsf{PBS}}]$.

```
$\mathsf{BLD}^{\mathsf{A},b}_{\mathsf{PBSch}}(\lambda)$, $\boxed{\mathsf{G}_1^{\mathsf{A},b}}$, $\boxed{\mathsf{G}_2^{\mathsf{A},b}}$
$(p, \mathbb{G}, G, \mathsf{H}) = sp \leftarrow \mathsf{Sch.Setup}(1^\lambda)$
$(crs, \tau) \leftarrow \mathsf{NArg.Setup}(sp)$
$(ek, dk) \leftarrow \mathsf{PKE.KeyGen}(1^\lambda)$
$(prd_0, prd_1, m_0, m_1, X, state) \leftarrow \mathsf{A}_1(crs, ek)$
if $(\exists\, i, j \in \{0,1\} : \mathsf{P}(prd_i, m_j) = 0)$ :
    return $0$
$(sess_0, sess_1) \coloneqq (\mathtt{init}, \mathtt{init})$
// sample randomness used by $\textsc{User}_0(\cdot)$:
$(\alpha_0, \alpha_1, \beta_0, \beta_1) \leftarrow\$ \mathbb{Z}_p^4$
$(\rho_0, \rho_1) \leftarrow\$ \mathcal{R}_{ek}^2$
$b' \leftarrow \mathsf{A}_2^{\textsc{User}_0, \textsc{User}_1, \textsc{User}_2}(state)$
return $b'$
```

```
$\textsc{User}_0(i)$
if $sess_i \neq \mathtt{init}$ : return $\bot$
$sess_i = \mathtt{open}$
$M \coloneqq (m_{i \oplus b}, \alpha_i, \beta_i)$
  ┌ // encrypt fixed vector instead:
  └ $M \coloneqq (\bar{m}, 0, 0)$  // $\bar{m}$ ...arbitrary fixed msg
$C_i \coloneqq \mathsf{PKE.Enc}(ek, M; \rho_i)$
return $C_i$
```

```
$\textsc{User}_1(i, R)$
if $sess_i \neq \mathtt{open}$ : return $\bot$
$sess_i = \mathtt{await}$
$R_i = R$
$R'_{i \oplus b} = R_i + \alpha_i G + \beta_i X$
$c_i = (\mathsf{H}(R'_{i \oplus b}, X, m_{i \oplus b}) + \beta_i) \bmod p$
$\theta \coloneqq (X, R_{i \oplus b}, c_i, C_i, prd_i, ek)$
$w \coloneqq (m_{i \oplus b}, \alpha_i, \beta_i, \rho_i)$
$\pi \leftarrow \mathsf{NArg.Prove}(crs, \theta, w)$
  ┌ // simulate the proof instead
  └ $\pi \leftarrow \mathsf{NArg.SimProve}(crs, \tau, \theta)$
return $(c_i, \pi)$
```

```
$\textsc{User}_2(i, s)$
if $sess_i \neq \mathtt{await}$ : return $\bot$
$sess_i = \mathtt{closed}$
if $sG = R_i + c_i X$ :
    $\sigma_{i \oplus b} = \big(R'_{i \oplus b}, (s + \alpha_i) \bmod p\big)$
else : $\sigma_{i \oplus b} = \bot$
if $(sess_0 = sess_1 = \mathtt{closed})$ :
    if $(\sigma_0 = \bot \vee \sigma_1 = \bot)$ :
        return $(\bot, \bot)$
    return $(\sigma_0, \sigma_1)$
return $\varepsilon$
```

Fig. 9. The blindness game from Figure 3 for the scheme $\mathsf{PBSch}[\mathsf{P}, \mathsf{GrGen}, \mathsf{HGen}, \mathsf{PKE}, \mathsf{NArg}]$ from Figure 4 (ignoring all boxes) and hybrid games used in the proof of Theorem 2. G_1 includes the light green box and G_2 includes both boxes.

According to the definition of game ZK, Z_b receives as input *crs* generated by $\mathsf{NArg.Setup}$ on input *sp* generated by $\mathsf{NArg.Rel}$, which is defined as $\mathsf{Sch.Setup}$. With this, Z_b simulates the game BLD^b for A, using its oracle $\textsc{Prove}$ to generate the proofs π required to answer A's queries to $\textsc{User}_1$.

When A_2 outputs its decision bit b', Z_b returns b' to its challenger. By the definition of Z_b for $b \in \{0,1\}$, we have $\Pr\big[\mathsf{ZK}^{\mathsf{Z}_b,0}_{\mathsf{NArg}[\mathsf{R}]}\big] = \Pr\big[\mathsf{BLD}^{\mathsf{A},b}_{\mathsf{PBSch}}\big]$, and $\Pr\big[\mathsf{ZK}^{\mathsf{Z}_b,1}_{\mathsf{NArg}[\mathsf{R}]}\big] = \Pr\big[\mathsf{G}_1^{\mathsf{A},b}\big]$ and therefore

$$\mathsf{Adv}^{\mathsf{ZK}}_{\mathsf{NArg}[\mathsf{R}_{\mathsf{PBS}}],\mathsf{Z}_b} \coloneqq \big|\Pr\big[\mathsf{ZK}^{\mathsf{Z}_b,0}_{\mathsf{NArg}[\mathsf{R}_{\mathsf{PBS}}]}\big] - \Pr\big[\mathsf{ZK}^{\mathsf{Z}_b,1}_{\mathsf{NArg}[\mathsf{R}_{\mathsf{PBS}}]}\big]\big| = \big|\Pr\big[\mathsf{BLD}^{\mathsf{A},b}_{\mathsf{PBSch}}\big] - \Pr\big[\mathsf{G}_1^{\mathsf{A},b}\big]\big| .$$

```
$\mathsf{Z}_b^{\textsc{Prove}}(crs)$
$(p, \mathbb{G}, G, \mathsf{H}) :\subseteq crs$
$(ek, dk) \leftarrow \mathsf{PKE.KeyGen}(1^\lambda)$
$(prd_0, prd_1, m_0, m_1, X, state) \leftarrow \mathsf{A}_1(crs, ek)$
if $\exists\, i, j \in \{0,1\} : \mathsf{P}(prd_i, m_j) = 0$ :
    return $0$
$(sess_0, sess_1) := (\mathtt{init}, \mathtt{init})$
$(\alpha_0, \alpha_1, \beta_0, \beta_1) \leftarrow\$ \mathbb{Z}_p^4$
$(\rho_0, \rho_1) \leftarrow\$ \mathcal{R}_{ek}^2$
$b' \leftarrow \mathsf{A}_2^{\textsc{User}_0, \textsc{User}_1, \textsc{User}_2}(state)$
return $b'$
```

```
$\textsc{User}_1(i, R)$
if $sess_i \neq \mathtt{open}$ : return $\perp$
$sess_i = \mathtt{await}$
$R_i = R$
$R'_{i\oplus b} = R_i + \alpha_i G + \beta_i X$
$c_i = (\mathsf{H}(R'_{i\oplus b}, X, m_{i\oplus b}) + \beta_i) \bmod p$
$\theta := (X, R_{i\oplus b}, c_i, C_i, prd_i, ek)$
$w := (m_{i\oplus b}, \alpha_i, \beta_i, \rho_i)$
$\pi \leftarrow \textsc{Prove}(\theta, w)$
return $(c_i, \pi)$
```

Fig. 10. Z_b playing against zero-knowledge of the $\mathsf{NArg}[\mathsf{R}_{\mathsf{PBS}}]$. The oracles $\textsc{User}_0$ and $\textsc{User}_2$ simulated to A_2 are as defined in game G_0 in Figure 9.

Together with the triangular inequality this yields:

$$\begin{aligned}\mathsf{Adv}^{\mathsf{BLD}}_{\mathsf{PBSch},\mathsf{A}} &:= \left|\Pr\left[\mathsf{BLD}^{\mathsf{A},1}_{\mathsf{PBSch}}\right] - \Pr\left[\mathsf{BLD}^{\mathsf{A},0}_{\mathsf{PBSch}}\right]\right| \\ &= \left|\Pr\left[\mathsf{BLD}^{\mathsf{A},1}_{\mathsf{PBSch}}\right] - \Pr\left[\mathsf{G}_1^{\mathsf{A},1}\right] + \Pr\left[\mathsf{G}_1^{\mathsf{A},1}\right] - \Pr\left[\mathsf{BLD}^{\mathsf{A},0}_{\mathsf{PBSch}}\right] + \Pr\left[\mathsf{G}_1^{\mathsf{A},0}\right] - \Pr\left[\mathsf{G}_1^{\mathsf{A},0}\right]\right| \\ &\leq \mathsf{Adv}^{\mathsf{ZK}}_{\mathsf{NArg}[\mathsf{R}_{\mathsf{PBS}}],\mathsf{Z}_1} + \left|\Pr\left[\mathsf{G}_1^{\mathsf{A},1}\right] - \Pr\left[\mathsf{G}_1^{\mathsf{A},0}\right]\right| + \mathsf{Adv}^{\mathsf{ZK}}_{\mathsf{NArg}[\mathsf{R}_{\mathsf{PBS}}],\mathsf{Z}_0}\ . \qquad (15)\end{aligned}$$

$\underline{\mathsf{G}_2}$. In G_2 we modify the $\textsc{User}_0$ and encrypt an arbitrary fixed message $\bar{m} \in \mathcal{M}_{sp}$ and $(0,0)$ instead of α and β. To show that this only changes A's behavior in a negligible way, in Figure 11 we define adversaries C_0 and C_1 playing in game CPA for scheme PKE.

```
$\mathsf{C}_b^{\textsc{Enc}}(ek)$
$1^\lambda :\subseteq ek$
$(p, \mathbb{G}, G, \mathsf{H}) = sp \leftarrow \mathsf{Sch.Setup}(1^\lambda)$
$(crs, \tau) \leftarrow \mathsf{NArg.Setup}(sp)$
$(prd_0, prd_1, m_0, m_1, X, state) \leftarrow \mathsf{A}_1(crs, ek)$
if $\exists\, i, j \in \{0,1\} : \mathsf{P}(prd_i, m_j) = 0$ :
    return $0$
$(sess_0, sess_1) := (\mathtt{init}, \mathtt{init})$
$(\alpha_0, \alpha_1, \beta_0, \beta_1) \leftarrow\$ \mathbb{Z}_p^4$
$b' \leftarrow \mathsf{A}_2^{\textsc{User}_0, \textsc{User}_1, \textsc{User}_2}(state)$
return $b'$
```

```
$\textsc{User}_0(i)$
if $sess_i \neq \mathtt{init}$ : return $\perp$
$sess_i = \mathtt{open}$
$M_0 := (\bar{m}, 0, 0)$
$M_1 := (m_{i\oplus b}, \alpha_i, \beta_i)$
$C_i \leftarrow \textsc{Enc}(M_0, M_1)$
return $C_i$
```

Fig. 11. C_b playing against CPA security of PKE. The oracles $\textsc{User}_1$ and $\textsc{User}_2$ simulated to A_2 are defined as in game G_1 in Figure 9.

By the definition of game CPA, C_b, for $b \in \{0,1\}$, gets as input the encryption key ek, from which it reads out the security parameter 1^λ, uses it to generate the parameters $p, \mathbb{G}, G, \mathsf{H}$ and crs and simulates $\mathsf{G}_1^{\mathsf{A},b}$ to A. During a call of $\textsc{User}_0(i)$, C_b sets $M_0 := (\bar{m}, 0, 0)$ and $M_1 := (m_{i \oplus b}, \alpha_i, \beta_i)$, calls its encryption oracle on (M_0, M_1) and sends the received ciphertext C_i to A_2. (Note that the randomness used to generate C_i is not known to C_b, but since the proofs in $\textsc{User}_1$ are simulated, the witness containing this randomness is no longer required.)

By construction of C_b we have $\Pr[\mathsf{CPA}_{\mathsf{PKE}}^{\mathsf{C}_b,0}] = \Pr[\mathsf{G}_2^{\mathsf{A},b}]$ and $\Pr[\mathsf{CPA}_{\mathsf{PKE}}^{\mathsf{C}_b,1}] = \Pr[\mathsf{G}_1^{\mathsf{A},b}]$ for $b \in \{0,1\}$, and hence

$$\mathsf{Adv}_{\mathsf{PKE},\mathsf{C}_b}^{\mathsf{CPA}} := \left|\Pr[\mathsf{CPA}_{\mathsf{PKE}}^{\mathsf{C}_b,1}] - \Pr[\mathsf{CPA}_{\mathsf{PKE}}^{\mathsf{C}_b,0}]\right| = \left|\Pr[\mathsf{G}_1^{\mathsf{A},b}] - \Pr[\mathsf{G}_2^{\mathsf{A},b}]\right| \quad \text{for } b \in \{0,1\} \; .$$

Together with the triangular inequality, this yields:

$$\begin{aligned} \left|\Pr[\mathsf{G}_1^{\mathsf{A},1}] - \Pr[\mathsf{G}_1^{\mathsf{A},0}]\right| &= \left|\Pr[\mathsf{G}_1^{\mathsf{A},1}] - \Pr[\mathsf{G}_2^{\mathsf{A},1}] + \Pr[\mathsf{G}_2^{\mathsf{A},1}] - \Pr[\mathsf{G}_2^{\mathsf{A},0}] + \Pr[\mathsf{G}_2^{\mathsf{A},0}] - \Pr[\mathsf{G}_1^{\mathsf{A},0}]\right| \\ &\le \mathsf{Adv}_{\mathsf{PKE},\mathsf{C}_1}^{\mathsf{CPA}} + \left|\Pr[\mathsf{G}_2^{\mathsf{A},1}] - \Pr[\mathsf{G}_2^{\mathsf{A},0}]\right| + \mathsf{Adv}_{\mathsf{PKE},\mathsf{C}_0}^{\mathsf{CPA}} \; . \end{aligned} \tag{16}$$

Reducing G_2 to perfect blindness of "plain" Blind Schnorr. The signer's view after the successful completion of the two signing sessions consists of the parameters (crs, ek) and the signatures with the corresponding messages: $(m_0, (R'_0, s'_0))$ and $(m_1, (R'_1, s'_1))$, as well as $\{(C_i, R_i, c_i, \pi_i, s_i,)_{i \in \{0,1\}}\}$ 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, and the argument π_i is simulated, they hold no information on bit b. Now take $(m_j, (R'_j, s'_j))$ for $j \in \{0,1\}$ and assume it corresponds to session i with (R_i, c_i, s_i). Fix $\alpha := s'_j - s_i$. Now there exists exactly one β s.t. $R'_j = R_i + \alpha G + \beta X$. This means, that both session tuples (R_0, c_0, s_0) and (R_1, c_1, s_1) explain (R'_j, s'_j). Hence the advantage in distinguishing G_2 with $b = 0$ from G_2 with $b = 1$ is

$$\left|\Pr[\mathsf{G}_2^{\mathsf{A},1}] - \Pr[\mathsf{G}_2^{\mathsf{A},0}]\right| = 0 \; .$$

This, together with (15) and (16), concludes the proof. □