(Concurrently Secure) Blind Schnorr Signatures

Georg Fuchsbauer and Mathias Wolf

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

23 April 2023

Abstract. Many applications of blind signatures (notably in blockchains) require the resulting signatures to be compatible with the existing system. This makes schemes that produce Schnorr signatures (now being standardized and supported by major cryptocurrencies like 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 practical, 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 introduce the notion of predicate blind signatures, in which the signer can define a predicate that the blindly signed message must satisfy.

We provide proof-of-concept implementations and benchmarks for various choices of primitives and scenarios, including blindly signing Bitcoin transactions conditioned on certain properties.

Keywords: Schnorr signatures · (partially) blind signatures · concurrent security · implementation · Bitcoin

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, BFQ21], they have also become a central primitive for e-voting protocols [Cha88, FOO93, Her97, ROG07] and anonymous credentials [Bra94, CL01, CL04, CG08, FP09, BCC+09, Fuc11, BL13, FHS15].

Recently, blind signatures have seen a renewed interest due to their applicability in privacy-sensitive settings ranging from *COVID-19 contact-tracing* applications [BRS20, DLZ+20] to *advanced VPNs* [Goo], *private relays* [App] and *private access tokens* [HIP+]. In the context of blockchains, blind signatures have been considered for increasing on-chain privacy, e.g. via *blindly signing contracts*, *blind coin swaps* or *trustless tumbler services* [HBG16, HAB+17, Nic19, LLL+19].

While blind-signing protocols that yield signatures of a standardized scheme are desirable in general, this can also be a stringent requirement: changing the supported signature schemes for blockchain systems requires consensus, which is a lengthy process. For example, blind coin swaps using *scriptless scripts* [Nic19] in Bitcoin require blind issuing of Schnorr signatures.

Schnorr signatures. One of the most relevant signature schemes today are Schnorr signatures [Sch90]. Since their patent expired in 2008, they are outpacing RSA signatures in

application counts. Schnorr signatures are much smaller, more efficiently verifiable for a comparable security level and less error-prone in implementations due to fewer edge cases. (EC)DSA, a NIST-standardized signature scheme, has efficiency comparable to Schnorr, but it requires unrealistic idealizations to be proved secure [FKP16, 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]

Schnorr signatures are now supported by major blockchain systems such as *Bitcoin* [WNR20], *Bitcoin Cash*, *Litecoin* or *Polkadot*, and, in the form of EdDSA (and other variants), in *Monero*, *Zcash*, or *Cardano*. Their adoption was also motivated by the privacy and scalability improvements [BDN18, MPSW19, BK22] they enable, properties which the *Mimblewimble* protocols [FOS19, FO22] crucially rely on.

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 be insecure when the signer runs several signing sessions simultaneously. In applications, concurrent security (where the adversary might interweave several signing sessions) is essential, as otherwise the signer can only engage in a signing session once the previous session has been finished or canceled. Protocols that only allow sequential execution are thus vulnerable to denial-of-service (DoS) attacks, such as simple connection time-outs, and are therefore severely limited in throughput range. This motivated the development of concurrently secure blind signature schemes [Bol03, BNPS03, Oka06, KZ06, HKKL07, KLR21, KLX22, CHL+22, TZ22, HLW22], or even *round-optimal* schemes [Fis06, AFG+10, GRS+11, GG14, FHKS16, Gha17], in which the user and the signer both only send a single message and which thus provide concurrent security by default.

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. While Wagner's subexponential-time attack [Wag02] had showed that ROS was not as hard as conjectured, more recently Benhamouda et al. [BLL+21] presented a polynomial-time algorithm. They show how attackers that open polynomially many signing sessions (concretely, 256, if that is the security parameter) can efficiently forge signatures (concretely, derive 257 signatures on messages of their 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 blind-signing sessions of which the signer finishes only one picked at random. They prove unforgeability in the algebraic group model and the ROM from

[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, $Z = \sum \zeta_i X_i$.

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" (mROS) is infeasible. While mROS appears harder than ROS, it can be solved in subexponential time, which decreases the security of their scheme to 70-bit for standard instantiations of Schnorr.

The question whether the original blind Schnorr signature scheme [CP93] 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. Katz, Loss and Rosenberg [KLR21] extend a technique [Poi98] which applied to blind Schnorr [CP93, KLX22] yields a concurrently secure scheme. However, the resulting signatures are not Schnorr signatures.[3]

Garg et al. [GRS+11] construct a round-optimal (and thus concurrently secure) blind-signing protocol for any signature scheme. They "stress that [their] result is only a feasibility result", as it is based on Yao's garbled circuits and makes use of complexity leveraging.

The lack of practical 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 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 note [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.

The state of the art in concurrently secure (partially) blind signatures schemes are "Schnorr-like" schemes (which do not require pairings) on the one hand: Tessaro and Zhu [TZ22] build on blind Schnorr [FPS20] and obtain a scheme with signatures in $\mathbb{Z}_p^4$ and a proof from DL in the AGM+ROM (where p is the order of the underlying group $\mathbb{G}$); Barreto and Zanon [BZ23] achieve blindness by replacing parts of the Schnorr signature by a Schnorr proof of knowledge of it; their scheme has signatures in $\mathbb{G} \times \mathbb{Z}_p^2$ and a proof from OMDL in the ROM.

One the other hand, Hanzlik, Loss and Wagner [HLW22] improve on recent techniques [KLR21, CHL+22] for a pairing-based scheme [Bol03] proven from (co-)CDH in the ROM, with round-optimal issuing [Fis06] but relatively large signatures.[4]

Our contributions

We present the first concurrently secure blind and partially blind signing protocol for issuing standard Schnorr signatures with rigorous security guarantees. Our blind-signing protocol

[3] Blind signing is a 7-move protocol and, due to a loose security proof, 12 000-bit groups would be needed [CHL+22]. The communication cost per signing session, which is linear in the number of preceding sessions, was then reduced to logarithmic [CHL+22] (but no Schnorr-based variants are mentioned).

[4] A signature in [HLW22] consists of a BLS signature [BLS01], as well as $K-1$ keys and commitment openings. For 128-bit security the authors suggest $K = 33$, which yields 5.71 kB per signature, compared to 128 bytes for [TZ22].

consists of two rounds (four moves). In contrast to the only prior practical scheme [FPS20] (which relies on an unstudied assumption), our scheme can be used for Schnorr instantiations over 256-bit elliptic curves, yielding signatures of size 64 bytes.[5]

Overview of our scheme. Our starting point is the plain protocol [CP93], against which the recent forgery 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. For each session, the adversary obtains the signer's first protocol message, a group element R. For each session, it then samples two possible sets of "blinding values" (which 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.

An attempt to prevent this specific attack could be to oblige the user to commit to her secrets (blinding values and the message to be signed) *before* receiving the value R. In the second round, the user must then prove 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 the concrete attack [BLL+21], but to make the scheme unforgeable under concurrent signing sessions, as we show by giving a security proof.[6]

Note 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 (predicate) about this message. Our construction therefore naturally 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[7] 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 is a non-standard 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

[5] Assuming 128-bit security for DL on the curve and n-bit security for Schnorr signatures and NIZK soundness, Theorem 1 yields n-bit security as long as $n \geq 126 - \log q$, where q is the number of signing session an attacker *closes* successfully. In contrast, for 256-bit curves, *clause blind Schnorr* [FPS20] only achieves 70-bit security due to attacks on mROS (cf. [TZ22]).

[6] Having the user commit to her randomness upfront and later prove that her protocol messages are consistent with it has been used in previous blind signature constructions via cut-and-choose [Poi98, KLR21, CHL+22, HLW22]. However, if the user revealed all of her secrets (in the "chosen" sessions), this would break blindness; in these protocols the signer therefore signs a (hiding) *commitment* to the actual message (and hence the resulting signatures need to contain the commitment opening).

[7] This allows us to extract the committed values during the signing queries in our proof of unforgeability in a "straight-line" fashion. We cannot use commitments (which could be more efficient) that assume non-blackbox extraction, because the reduction would have to run extractors that run other extractors, which would lead to an exponential blow-up of its running time. (See Appendix D for a detailed discussion.) Moreover, a proof of knowledge would also not help since we need to extract before the proven statement is known.

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.[8] Assuming unforgeability of (Schnorr) signatures when proving the security of a protocol built on top has recently been done in the context of multi- and threshold signatures [CKM21, BCK+22].

The security of our scheme thus relies solely on the security of the used building blocks and we do not make any additional assumptions, such as OMDL or (variants of) ROS, or work in idealized models.

Avoiding a trusted setup. For the sake of generality, our security notions assume trusted parameters (which is necessary for NIZKs in the standard model [GO94]). However, depending on the instantiation of NArg and PKE, a trusted setup can easily be avoided in practice (and formally, one would rely on the random oracle model): When instantiating PKE e.g. with ElGamal [ElG85] over an elliptic-curve group (as we do in our implementations), one can generate a public key for which no one knows the secret key by "hashing into the curve" [BF01, BCI+10, WB19] (and model the hash function as a random oracle).

For the NIZK, one can use a scheme that is secure in the ROM or requires a "uniform reference string" (which could also be created via a hash function modeled as a random oracle) [BBB+18, BCR+19, BFS19, BGH19, KPV19, BBHR19, COS20, Set20, CHM+20, SL20, Com21, Zer22, CBBZ22].

If the signer sets up the NIZK parameters (and can thus be sure that no one knows a 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 our scheme, which protects against malicious signers, would then still hold. *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. The users would have to perform this check once.[9] In practice, they could optimistically trust the signer, since the discovery of the inconsistency of her parameters would harm her reputation.

A third possibility is to accept trusted parameters, but use a scheme that has "universal" parameters [GKM+18], such as *Plonk* [GWC19] and *Marlin* [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). As the main efficiency bottleneck of our construction is the NIZK proof system, we give two prototype implementations using *Groth16* and *Plonk* (see below).

EdDSA. Since EdDSA [BDL+12] is based on Schnorr signatures, our construction also yields (predicate) blind EdDSA signatures. EdDSA, and other types of Schnorr signatures [WNR20] derive the randomness $r = \log R$ used during signing deterministically, by hashing the message and the signer's secret key.[10] This 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

[8] This is also why we do not use the random oracle model for extractable commitments (but rely on PKE instead), in contrast to other work [KLR21].

[9] We analyze the computational complexity of a probabilistic optimization of this check in Appendix F.

[10] If the same r was used for different messages, the secret key would be leaked, which is thereby prevented; resilience to side-channel attacks is also increased [NS02].

privacy guarantees. While in partially blind signatures, the signer and the user agree on the public part of the messages 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. PBS easily generalize to predicates that take a witness as additional input. This allows to enforce NP-statements on the message during blind signing.

Applications. Predicate blind signatures address conditional privacy-preserving authorization in a general way. For example, a payment provider may only authorize transactions compliant with the law and/or internal rules, but otherwise protect customer privacy. For example, transactions to certain countries can only have a limited amount. Using PBS, the provider neither learns the amount nor the recipient's location, but that the criteria are met.

Trying to realize this 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 the message conforms to its public part. Worse, the signature would reveal the conditions, which remain hidden when using PBS.

We present a (concurrently secure) instantiation of PBS whose signatures are standard Schnorr signatures. As these are supported by Bitcoin [WNR20], this now enables concurrently secure blind coin swaps [Nic19]. But using the "predicate" functionality opens up new applications. For example, PBS can add anonymity to payments by users that entrust their coins to a cryptocurrency exchange: The user would construct a payment from the exchange's address and have it blindly signed using a predicate that enforces (an upper-bound of) the paid amount. The exchange can then debit the user's account by that amount.

Implementations

To give rough estimates of the efficiency of our construction, we implemented its computationally heavy part, the proof system NArg. We consider blind, partially blind and predicate-blind settings, in particular the conditional blind signing of Bitcoin transactions mentioned above.

We consider two choices for NArg: (G) *Groth16* [Gro16] (*Iden3* implementation [ide]), a zk-SNARK with trusted parameters (for soundness but subversion-zero-knowledge), and (P) the "universal" zk-SNARK *PlonK* [GWC19] (*Fluidex* implementation [Plo]). For both proof systems the relation to be proven is represented as an arithmetic circuit over the field $\mathbb{F}_p$. We wrote the circuits for the various scenarios in the domain-specific language of Circom 2.0 [ide] and made them publicly available [mot]. Our benchmarks were conducted on a standard laptop as a proof of concept.

We instantiate the encryption scheme PKE using the DHIES [ABR98] KEM over the *Baby JubJub* (BJJ) curve group [BJJ20] and a sponge hash from the Poseidon family [GKR+21]. As DEM we use the one-time pad in $\mathbb{F}_p$. (These choices are motived by their concise circuit representations)

We first consider (optimized) scenarios in which the underlying Schnorr instantiation also uses BJJ and Poseidon. For both proof systems (G) and (P), the *proving key* (i.e., the part of the CRS used by the user requesting a blind signature) is around 2 MB long; computation of proofs takes under one (G) or two (P) seconds; proofs are 402 bytes (G) or around 800 bytes (P) and take under half a second to verify (see rows (A1)–(A3) in Table 1, page 25).

We then consider the scenario of deploying our scheme for Bitcoin, that is, considering as Schnorr parameters the curve secp256k1 and SHA-256 [WNR20]. These are not efficiently "arithmetizable" in our proof-system configurations (we implement both (G) and (P) over the BN254 curve [BN06], whose order is incompatible with the secp256k1 base field). Performance is thus worse, but still practical. The CRS is now 550 MB and proofs take around 1 minute (G) or 3 minutes (P) to generate, while their size does not increase (G) or by 150 bytes (P); verification time also remains unchanged compared to the optimized scenario. There is thus little computational burden on the signer's side, like the cryptocurrency exchange in the above example.[11]

We also discuss possible approaches for constructing a proof system that interoperates well with the secp256k1 curve in Appendix G. To give a very rough idea of the potential efficiency, we consider (G) and (P) for BJB, thus a "compatible" curve, but maintain SHA-256. In this setting, the proving key reduces to less than 63 MB for both (G) and (P), and proofs take 5 (G) or 34 (P) seconds to compute. Finally, we give estimates for the running time of the user's check of the *Groth16* parameters when not trusting them. These can range from under a second up to five hours depending on the scenario. However, the user only needs to do this once (or trust someone else has done it).

Privacy-preserving applications that require minutes of calculation are not new in the blockchain space, as this was how long it took to compute "private" transactions in the first generation of *Zcash*. We emphasize that the numbers we obtained should be viewed as upper bounds, as the implementations are far from optimized and numbers depend on machine specifics (e.g., proof verification for (P) has been reported [Bot] to be 100 times faster than measured by us).

Comparison to 2PC. Blind signing for any scheme can be achieved by using generic two-party computation between the signer and the user [JLO97] (which would in general not yield concurrent security [HKKL07]). However, the resulting efficiency is several magnitudes worse than our approach. Jayaraman, Li and Evans [JLE17] use garbled circuits [Yao82] to implement two-party signing for ECDSA (of complexity comparable to Schnorr) over secp192k1 (thus smaller parameters than ours). Their variant providing security against malicious parties runs for around a day and requires 819 GB of data transfer per signing.

2 Preliminaries

2.1 Notation

For $n \in \mathbb{N}^+$ we denote by $[n]$ the set $\{1, \ldots, n\}$. 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 clear from the context, it 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 $x \coloneqq (a, \ldots, z)$ and $x[i]$ denotes the i-th element, which we set to ε if it does not exist.

[11] We expect implementations for Schnorr instantiated over ed25519 (Circom 2.0 implementation of [EL]), i.e., blind signing of EdDSA [BDL+12] to yield similar benchmarks using BN254, since the base field of this curve is also incompatible with the BN254 scalar field.

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.) in the security parameter 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 output 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 $b_1, b_2, \ldots$ (that is, these can be parsed or obtained from a in polynomial time), we write $(b_1, b_2, \ldots) :\subseteq a$. We shorten $a \equiv b \pmod q$ to $a \equiv_q b$.

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 $(q, \mathbb{G}, G)$, where $\mathbb{G}$ is the description of a group of prime order q s.t. $\lceil \log_2(q) \rceil = \lambda$, 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*

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

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

$\mathsf{DL}^{\mathsf{A}}_{\mathsf{GrGen}}(\lambda)$

$(q, \mathbb{G}, G) \leftarrow \mathsf{GrGen}(1^\lambda)$
$x \leftarrow\!\!\$\, \mathbb{Z}_q\,;\ X \coloneqq xG$
$y \leftarrow \mathsf{A}(q, \mathbb{G}, G, X)$
return $(y = x)$

2.3 Non-Interactive Zero-Knowledge Arguments

We define non-interactive zero-knowledge argument (NIZK) systems with respect to *parameterized relations* $\mathsf{R}\colon \{0,1\}^* \times \{0,1\}^* \times \{0,1\}^* \to \{0,1\}$, which are ternary relations that run in polynomial time in 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 $\mathcal{L}_{par_{\mathsf{R}}} \coloneqq \{\theta \mid \exists w : \mathsf{R}(par_{\mathsf{R}}, \theta, w) = 1\}$. A NIZK for a relation R is a tuple of efficient p.p.t. 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 *relation parameter generation algorithm*, on input the security parameter λ in unary, 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 an NP-language.
- $\mathsf{Setup}(par_{\mathsf{R}}) \to (crs, \tau)$: the *setup algorithm*, on input relation parameters par_{R}, returns a common reference string (CRS) crs and a simulation trapdoor τ; the CRS contains the description of par_{R}, i.e. $par_{\mathsf{R}} :\subseteq crs$.
- $\mathsf{Prove}(crs, \theta, w) \to \pi$: the *prover algorithm*, on input a CRS crs, a statement θ and a witness w, outputs a proof π.

– $\mathsf{Vfy}(crs, \theta, \pi) =: 0/1$: the deterministic p.t. *verification algorithm*, on input a CRS crs, a statement θ and a proof π, outputs 1 (accept) or 0 (reject).
– $\mathsf{SimProve}(crs, \tau, \theta) \to \pi$: the *simulation algorithm*, on input a CRS crs, a simulation trapdoor τ and a statement θ, outputs a proof π.

Definition 3. *A system* $\mathsf{NArg}[\mathsf{R}]$ *is* **(perfectly) correct** *if for every adversary* A *and* $\lambda \in \mathbb{N}$*:*

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

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

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

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

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

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

Definition 5. *A system* $\mathsf{NArg}[\mathsf{R}]$ *is* ***computationally zero-knowledge*** *if for every adversary* A

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

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

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

$par_{\mathsf{R}} \leftarrow \mathsf{NArg.Rel}(1^\lambda)$
$(crs, \tau) \leftarrow \mathsf{NArg.Setup}(par_{\mathsf{R}})$
$b' \leftarrow \mathsf{A}^{\textsc{Prove}}(crs)$
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 π_b

2.4 Public-Key Encryption

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

– $\mathsf{KeyGen}(1^\lambda) \to (ek, dk)$, on input the security parameter, outputs an encryption key ek and a decryption key dk, where ek defines the message space $\mathcal{M}_{ek}$, the randomness space $\mathcal{R}_{ek}$ and the ciphertext space $\mathcal{C}_{ek}$.
– $\mathsf{Enc}(ek, M; \rho) =: C$, on input an encryption key ek, a message M, randomness $\rho \in \mathcal{R}_{ek}$, outputs a ciphertext $C \in \mathcal{C}_{ek}$ if $M \in \mathcal{M}_{ek}$ and $\perp$ otherwise.

– $\mathsf{Dec}(dk, C) =: M$ is deterministic and 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 *is* ***(perfectly) correct***[12] *if for all* $\lambda \in \mathbb{N}$*:*

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

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:*

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

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

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

Definition 8. *A signature scheme* Sig *has* ***(perfect) correctness*** *if for all* $\lambda \in \mathbb{N}$*:*

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

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:*

[12] Since we require probability 1, perfect correctness holds for *all* messages in $\mathcal{M}_{ek}$.

$\mathsf{Sch.Setup}(1^\lambda)$

$(q, \mathbb{G}, G) \leftarrow \mathsf{GrGen}(1^\lambda)$
$\mathsf{H} \leftarrow \mathsf{HGen}(q)$
$sp \coloneqq (q, \mathbb{G}, G, \mathsf{H})$
return sp

$\mathsf{Sch.KeyGen}(sp)$

$(q, \mathbb{G}, G, \mathsf{H}) \coloneqq sp$
$x \leftarrow\!\!\$\, \mathbb{Z}_q\,;\ X \coloneqq xG$
$sk \coloneqq (sp, x)\,;\ vk \coloneqq (sp, X)$
return (sk, vk)

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

$(q, \mathbb{G}, G, \mathsf{H}, x) \coloneqq sk\,;\ r \leftarrow\!\!\$\, \mathbb{Z}_q\,;\ R \coloneqq rG$
$c \coloneqq \mathsf{H}(R, xG, m)\,;\ s \coloneqq (r + cx) \bmod q$
$\sigma \coloneqq (R, s)$
return σ

$\mathsf{Sch.Ver}(vk, m, \sigma)$

$(q, \mathbb{G}, G, \mathsf{H}, X) \coloneqq vk$
$(R, s) \coloneqq \sigma$
$c \coloneqq \mathsf{H}(R, X, m)$
return $(sG = R + cX)$

Fig. 1. The **Schnorr signature** scheme $\mathsf{Sch}[\mathsf{GrGen}, \mathsf{HGen}]$ with key-prefixing based on a group generator GrGen and hash generator HGen.

$\mathsf{sEUF\text{-}CMA}^{\mathsf{A}}_{\mathsf{Sig}}(\lambda)$

$sp \leftarrow \mathsf{Sig.Setup}(1^\lambda)$
$(sk, vk) \leftarrow \mathsf{Sig.KeyGen}(sp)\,;\ \mathcal{Q} \coloneqq \emptyset$
$(m^*, \sigma^*) \leftarrow \mathsf{A}^{\textsc{Sign}}(vk)$
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 σ

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 q, and it requires a hash function that maps into $\mathbb{Z}_q$, which we define as being generated as follows.

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 means that the verification key is prepended to the message when signing and verifying. It 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], with a tight security proof. These proofs are easily adapted to strong unforgeability of the key-prefixing variant, which (in the AGM+ROM) also readily follows from the discrete-logarithm assumption and key-prefixing Schnorr signatures being strongly simulation-extractable proofs of knowledge of discrete logarithms in the AGM+ROM [FO22].

We consider these results and the fact that, despite their wide use, no vulnerabilities have been found in Schnorr signatures as ample evidence for the following assumption, used 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), 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 anything about the signed message, except 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 prd or not.

A PBS scheme $\mathsf{PBS}[\mathsf{P}]$ for P is defined by the following algorithms. We focus on schemes with 2-round (i.e., 4-message) signing protocols for concreteness.

- $\mathsf{Setup}(1^\lambda) \to par$: the *setup algorithm*, on input the security parameter, outputs public parameters par, which define a message space $\mathcal{M}_{par}$.
- $\mathsf{KeyGen}(par) \to (sk, vk)$: the *key generation algorithm*, on input the parameters par, outputs a signing/verification key pair (sk, vk), which implicitly contain par, i.e., $vk = (par, key)$, thus $par :\subseteq vk$.
- $\langle \mathsf{Sign}(sk, prd), \mathsf{User}(vk, prd, m)\rangle \to (b, \sigma)$: 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:

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

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

We generalize the definitions for blind signatures [JLO97] and partially blind signatures [AO00] in the following.

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

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

(Strong) unforgeability. For blind signatures this notion states that after the completion of n signing sessions, the user cannot compute $n+1$ distinct signatures. For *partially* blind signatures, 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.

Generalizing this to predicate blind signatures is not straightforward as messages can satisfy many predicates (whereas messages only have one public part). We therefore 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 exists an assignment to *successfully closed* signing sessions, so that each message satisfies the predicate of the assigned session. In particular, there exists an injective mapping $f\colon [n] \to [\ell]$ where ℓ is the number of closed signing sessions, so that $\mathsf{P}(prd_{f(i)}, m_i^*) = 1$, where prd_j was the predicate for the j-th 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 that it only considers closed signing sessions and ignores unfinished sessions when checking whether an attack was not trivial.

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

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

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. To win, A must output a non-empty vector $(m_i^*, \sigma_i^*)_{i\in[n]}$ of distinct valid message/signature pairs; moreover, there must not exist an injective mapping from the messages (m_i^*) to the predicates (prd_j) used in successfully closed signing sessions (stored in the list $\vec{\textsc{Prd}}$), so that every message is mapped to a predicate it satisfies.[13]

Blindness. Blindness requires that whenever the signer gets to see one of its signatures, it cannot determine in which session the signature was generated, except that it must have been in a session with a predicate satisfied by the message. As a more general notion, we define blindness for schemes *with parameters*. This also covers instantiations without (or with "empty" parameters), as discussed in Section 5.1, and then yields the standard notion.

[13] We 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].

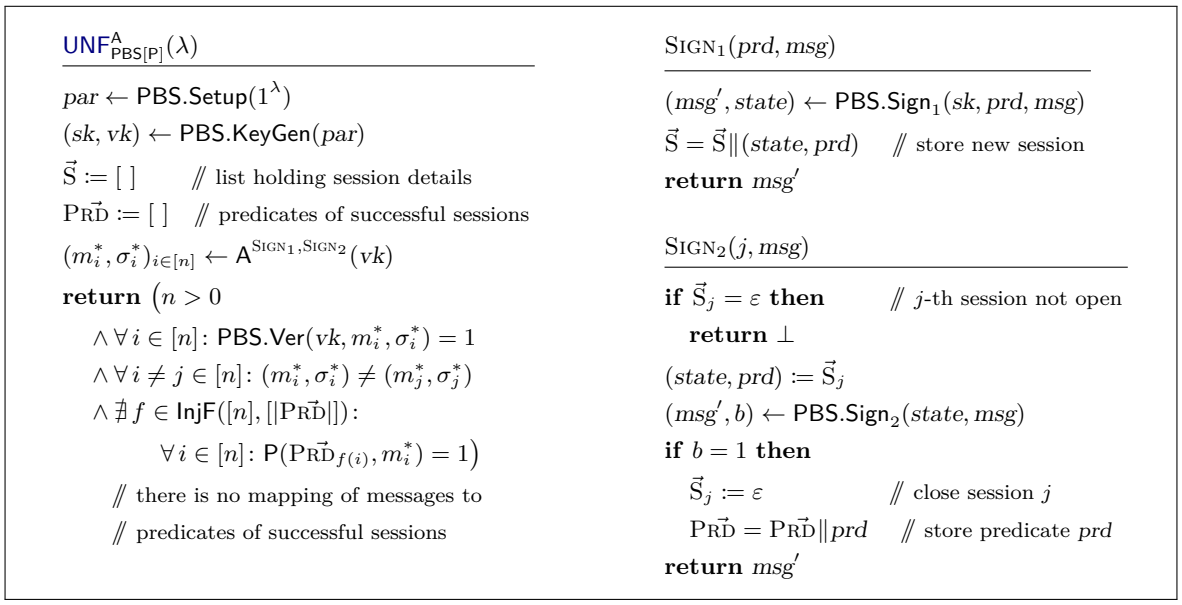

Fig. 2. The strong **unforgeability game** for a predicate blind signature scheme $\mathsf{PBS[P]}$ with a 2-round signing protocol. $\mathsf{InjF}(\mathcal{A}, \mathcal{B})$ denotes the set of all injective functions from set $\mathcal{A}$ to set $\mathcal{B}$. (Standard) unforgeability is obtained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

The adversary can choose its own verification key *key* (which together with the parameters constitutes vk), and, in addition to two messages m_0 and m_1, defines 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 protocol with the adversarial signer, 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

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

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

The adversary consists of two parts A_1 and A_2 of which A_1 outputs two messages m_0, m_1, predicates prd_0, prd_1, a verification key part *key* and a state *state*. The experiment only continues if both messages satisfy both predicates. It runs the signing protocol with the adversary acting as the user (modeled via three oracles $\mathrm{USER}_0, \mathrm{USER}_1, \mathrm{USER}_2$) in two concurrent sessions. The experiment asks for a blind signature on m_b for predicate prd_0 in the first session and on m_{1-b} for prd_1 in the second. If none of the obtained signatures σ_b and σ_{1-b} are $\perp$, 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. Blindness requires that no signer strategy is noticeably better than guessing the value of b.

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

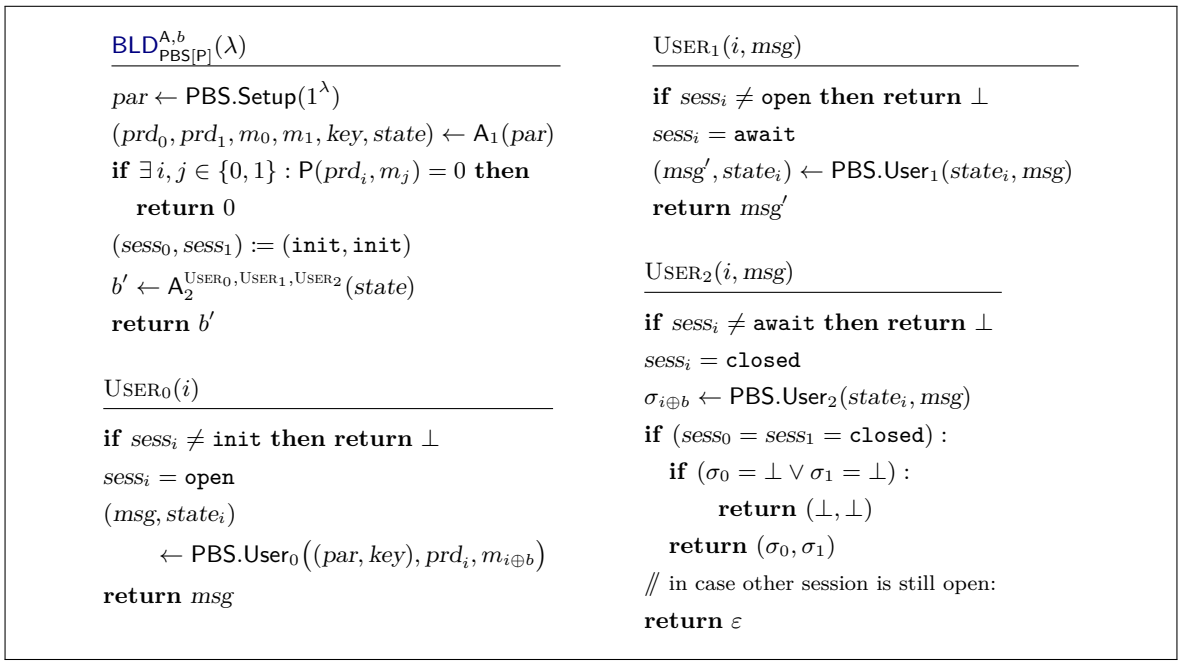

Fig. 3. The **blindness game** for a predicate blind signature scheme $\mathsf{PBS[P]}$ played by adversary $\mathsf{A} = (\mathsf{A}_1, \mathsf{A}_2)$. The operator "$\oplus$" is the XOR operation on bits, used to realize a swap of the message order if and only if $b = 1$.

Hiding the predicates. By allowing the adversary to output distinct predicates prd_0 and prd_1, our blindness notion yields an additional guarantee: that the resulting signature does not reveal anything about the predicate used in the signing session (apart from being satisfied by the message). 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 two blind signings of 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. This notion is implied by blindness (Definition 13) via a straightforward 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.

A partially blind signature scheme can be easily constructed 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:

$$\begin{array}{l} \mathsf{P}(prd, m) : \\ \hline (info, m') := m \\ \textbf{return } info = prd \end{array} \tag{1}$$

To issue a signature for *info* and secret part m', the signer and user run the PBS signing protocol for $prd := info$ and user input $m := (info, m')$. 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 (Definitions 12 and 13). To break unforgeability of a partially blind signature scheme, an adversary must output $(info, (m_i^*, \sigma_i^*)_{i\in[n]})$, for distinct pairs (m_i^*, σ_i^*) with $\mathsf{Vfy}(vk, info, m_i^*, \sigma_i^*) = 1$ for all $i \in [n]$, and the adversary queried the signing oracle $n-1$ times with public part $info$.

An adversary A against this unforgeability notion for our construction implies 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 $(info, (m_i', \sigma_i)_{i\in[n]})$, B returns $(m_i^* := (info, m_i'), \sigma_i)_{i\in[n]}$.

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 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 $\vec{\text{PRD}}$ (see Figure 2). For all $j \notin I$, we have $\mathsf{P}(\vec{\text{PRD}}_j, m_i^*) = 0$ (cf. (1), since $\vec{\text{PRD}}_j \neq info$ and $m_i^* = (info, m_i')$). Since $|I| \leq n-1$, there is no injective function f with $\mathsf{P}(\vec{\text{PRD}}_{f(i)}, m_i^*) = 1$ 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 their challenger samples the key pair (vk, sk) for the adversary, while our adversary can choose its own verification key part (yielding more realistic security guarantees). AO's adversary must output two pairs $(info_0, m_0')$ and $(info_1, m_1')$ with $info_0 = info_1$; our adversary must output two messages $(info_0, m_0')$ and $(info_1, m_1')$ and two predicates satisfied by both messages, which implies $info_0 = info_1$. The reduction generates the key pair for the AO adversary and then simply relays the oracle calls.

4 Predicate Blind Schnorr Signatures

4.1 Our Construction

Signature issuing in "plain" blind Schnorr signatures, which are not concurrently secure (Definition 12) [BLL+21], works as follows. Let $(q, \mathbb{G}, G)$ be the underlying group parameters and (x, X) be the signer's key pair. As with computing a Schnorr signature, the signer first samples $r \leftarrow\!\!\$\, \mathbb{Z}_q$, computes $R := rG$ and sends it to the user. The user samples two blinding values $(\alpha, \beta) \leftarrow\!\!\$\, \mathbb{Z}_q^2$ 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 as $c := (c' + \beta) \bmod q$ and sends c to the signer. The signer replies with $s := (r + cx) \bmod q$, which the user transforms to $s' := (s + \alpha) \bmod q$ and outputs the signature (R', s'). This is a valid Schnorr signature (Figure 1) since:

$$\begin{aligned} s'G = sG + \alpha G = (r + cx)G + \alpha G &= (r + (\mathsf{H}(R', X, m) + \beta)x)G + \alpha G \\ &= R + \alpha G + \beta X + \mathsf{H}(R', X, m)X \\ &= R' + \mathsf{H}(R', X, m)\, X\ . \end{aligned} \tag{2}$$

To make this protocol secure, we have the user first send an encryption C of the message m and the values α, β that he will use, before receiving the value R. 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 $\mathsf{R}_{\mathsf{Sch}}$:

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

$$\begin{array}{lll} R' \coloneqq R + \alpha G + \beta X & /\!/ \text{ blind the group element } R & \\ \textbf{return } \ c \equiv_q \mathsf{H}(R', X, m) + \beta & /\!/ \ c \text{ is computed from witness elements} & \\ \quad \wedge\ \mathsf{P}(prd, m) = 1 & /\!/ \ m \text{ satisfies the predicate } prd & \\ \quad \wedge\ \mathsf{PKE.Enc}(ek, (m, \alpha, \beta); \rho) = C & /\!/ \ C \text{ encrypts witness elements under } ek & \end{array} \tag{3}$$

This relation $\mathsf{R}_{\mathsf{Sch}}$ checks, for given parameters $(q, \mathbb{G}, G, \mathsf{H})$, 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 m satisfies the predicate prd; and whether the ciphertext C encrypts these values (m, α, β) using randomness ρ. As the parameters of $\mathsf{R}_{\mathsf{Sch}}$ are the Schnorr signature parameters, the relation-parameter sampling algorithm Rel for NArg is simply $\mathsf{Sch.Setup}$, i.e.,

$\mathsf{NArg.Rel}(1^\lambda)$

$(q, \mathbb{G}, G) \leftarrow \mathsf{GrGen}(1^\lambda)$
$\mathsf{H} \leftarrow \mathsf{HGen}(q)$
return $sp \coloneqq (q, \mathbb{G}, G, \mathsf{H})$

Let GrGen be a group generation algorithm and HGen be a hash function generator (which together define $\mathsf{Sch.Setup}$ as in Figure 1), let PKE be a public-key encryption scheme and P be a predicate compiler (which together define relation $\mathsf{R}_{\mathsf{Sch}}$), and let NArg be an argument system for $\mathsf{R}_{\mathsf{Sch}}$. Formalizing the ideas sketched above yields 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 $sp = (q, \mathbb{G}, G, \mathsf{H})$ 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 $\mathcal{M}_{par} \times \mathbb{Z}_q \times \mathbb{Z}_q \subseteq \mathcal{M}_{ek}$ for $par \coloneqq (crs, ek)$.

Correctness. Perfect correctness follows from perfect correctness of NArg and Eq. (2).

4.2 Security

Unforgeability. We bound the advantage in breaking the unforgeability (Definition 12) of PBSch by the advantages in breaking the security of the underlying primitives. Note that, in Assumption 1, we directly assume $\mathsf{sEUF\text{-}CMA}$ security of the Schnorr signature scheme. The reason is that all known security proofs of Schnorr signatures are in the random-oracle model [PS96, PS00, FPS20]; but the NArg relation $\mathsf{R}_{\mathsf{Sch}}$ in (3) depends on the used hash function, which would be replaced in the ROM by a random function, for which efficient proofs are not possible. 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.

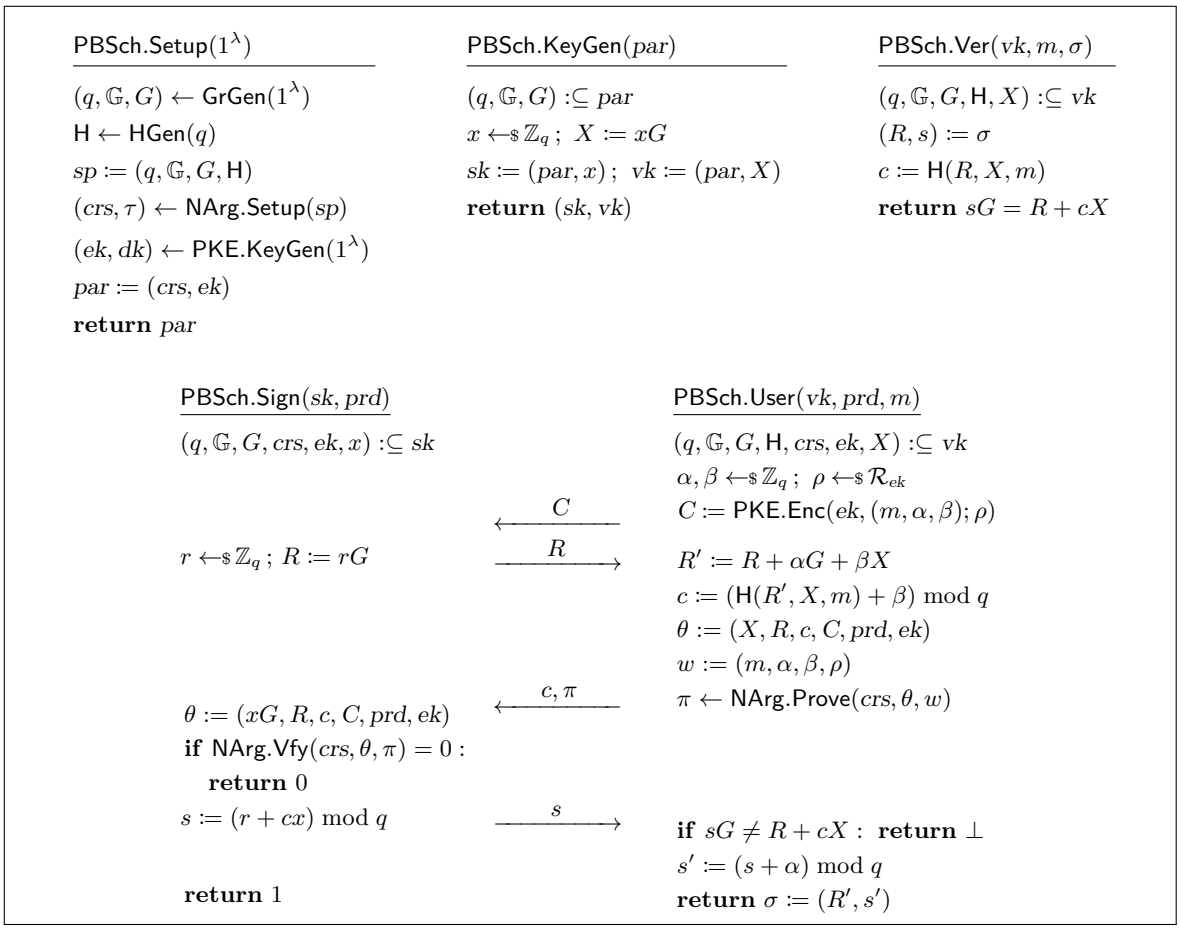

Fig. 4. The **predicate blind Schnorr signature** scheme$\mathsf{PBSch}[\mathsf{P}, \mathsf{GrGen}, \mathsf{HGen}, \mathsf{PKE}, \mathsf{NArg}]$ based on a predicate compiler P, a group generation algorithm GrGen, a hash generator HGen, a public-key encryption scheme PKE and non-interactive zero-knowledge argument scheme NArg for the relation $\mathsf{R}_{\mathsf{PBS}}$ from (3).

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* $\mathsf{NArg}[\mathsf{R}_{\mathsf{Sch}}]$ *be a non-interactive argument scheme for the relation* $\mathsf{R}_{\mathsf{Sch}}$ *from* (3)*. Then for any adversary* A *playing in game* UNF *against the PBS scheme* $\mathsf{PBSch}[\mathsf{P}, \mathsf{GrGen}, \mathsf{HGen}, \mathsf{PKE}, \mathsf{NArg}]$ *defined in Figure 4, successfully completing at most* q *sessions via the oracle* Sign_2*, there exist algorithms:*

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

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

$$\mathsf{Adv}^{\mathsf{UNF}}_{\mathsf{PBSch},\mathsf{A}}(\lambda) \leq \mathsf{Adv}^{\mathsf{sEUF\text{-}CMA}}_{\mathsf{Sch}[\mathsf{GrGen},\mathsf{HGen}],\mathsf{F}}(\lambda) + \mathsf{Adv}^{\mathsf{SND}}_{\mathsf{NArg}[\mathsf{R}_{\mathsf{Sch}}],\mathsf{S}}(\lambda) + q \cdot \exp(1) \cdot \mathsf{Adv}^{\mathsf{DL}}_{\mathsf{GrGen},\mathsf{D}}(\lambda)\ . \quad (4)$$

(Since unforgeability of Schnorr (tightly) implies DL, the security of the scheme follows from that of the underlying building blocks.)

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 X, the

reduction sets up *crs* and the encryption key *ek* and answers the adversary's signing queries. 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 $R := \bar{R} - \alpha G - \beta X$ 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 $\mathsf{R}_{\mathsf{Sch}}$, implies that $c \equiv_q \mathsf{H}(\bar{R}, X, m) + \beta$.

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

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. By perfect correctness of PKE, any abort can be used to break soundness of $\mathsf{NArg}[\mathsf{R}_{\mathsf{Sch}}]$. 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 "explaining" mapping of the adversary's messages to signing sessions, the adversary's output must contain a forged Schnorr signature.

Full tightness under a weaker unforgeability notion. If all *opened* sessions were considered when checking "non-triviality" in Definition 12, our scheme would be fully tight. This would mean that even when a signing session is not closed, a signer would consider this an issued signature. Formally, in Figure 2 the line $\text{PRD} = \text{PRD} \| prd$ would be included in SIGN_1 rather than SIGN_2.

In the proof of Theorem 1 we would not need to consider the case when an adversary completes a signature in a session it does not close, and there would be no reduction to DL (cf. Remark 1 in Appendix B). Instead of (4), we would get

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

Blindness. Perfect blindness of the "plain" blind Schnorr signature scheme is shown as follows [Sch01]: For every assignment of signature-issuing sessions to resulting message/signature pairs, 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, and thus satisfies Definition 13.

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* $\mathsf{NArg}[\mathsf{R}_{\mathsf{Sch}}]$ *be an argument system for the relation* $\mathsf{R}_{\mathsf{Sch}}$*. 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* $\mathsf{NArg}[\mathsf{R}_{\mathsf{Sch}}]$*, and*
- C_0 *and* C_1*, playing in game* CPA *against* PKE*,*

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

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

The proof of Theorem 2 can be found in Appendix C and proceeds via a sequence of game hops. Starting with game $\mathsf{BLD}^{\mathsf{A},b}_{\mathsf{PBS[P]}}$ 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 $\mathsf{NArg}[\mathsf{R}_{\mathsf{Sch}}]$ 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.

Alternative constructions. In Appendix D we discuss the necessity of straight-line extraction (as provided by the use of a PKE), which excludes the use of non-blackbox extractable commitments. We also argue why we cannot replace the proofs π by *zaps* [DN07], that is, witness-indistinguishable proofs without parameters.

4.3 Generalizing Predicates to NP-Relations

As a simple extension of our construction, we could allow P to take, in addition to a description $prd \in \{0,1\}^*$ and a message $m \in \{0,1\}^*$, a *witness* $w \in \{0,1\}^*$ attesting to m satisfying prd.

Model. The adaptations in the syntax and security definitions of $\mathsf{PBS[P]}$ are straightforward:
1) Algorithm $\mathsf{PBS.User}_0(vk, prd, m)$ takes an additional argument w.
2) In game BLD, the adversary A_1 additionally outputs w_0 and w_1 for which $\mathsf{P}(prd_i, m_j, w_j) = 1$ for all $i, j \in \{0,1\}$. In $\textsc{User}_0$, $\mathsf{PBS.User}_0$ takes additional argument $w_{i \oplus b}$.
3) When determining if A won game UNF, the check $\mathsf{P}(\vec{\mathrm{PRD}}_{f(i)}, m_i^*) = 1$ is replaced by

$$\exists\, w_i^* : \mathsf{P}(\vec{\mathrm{PRD}}_{f(i)}, m_i^*, w_i^*) = 1\ . \tag{5}$$

As for proof soundness, this might not be efficient. This could be remedied by an extractability-based definition, requiring that from an adversary against UNF one can extract witnesses $(w_i^*)_i$ that satisfy (5).

Construction. The only change in the construction is that the prd-witness for m is now included in the witness for $\mathsf{R}_{\mathsf{Sch}}$ in (3) and then used by P.

The proof of unforgeability only changes slightly. If we assume *knowledge soundness* of NArg, the reduction would extract the witness from π (in the hop from G_0 to G_1 in Appendix B) and the rest of the proof proceeds as before. When only relying on soundness of NArg the reduction would guess which proof π breaks soundness when G_1 aborts and the proof would thus incur a security loss in the number of $\textsc{Sign}_2$ calls.

5 Design Choices, Implementation Details and Benchmarks

5.1 Avoiding a Trusted Setup

When defining security 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 scheme PBSch (Figure 4), for a 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 are the protocol's "toxic waste" [COS20]. A party that knows τ can simulate proofs, and 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]: in the first round of signing, she commits to anything, and then simulates the proof (of a false statement) in the second round. On the other hand, a party that knows dk is able to decrypt the user's ciphertexts and thereby break blindness.

Consequently, we can neither let the signer nor the user run $\mathsf{PBSch.Setup}$. If the signer runs it, this breaks the user's security (blindness); if the user runs it, the signer's security (unforgeability) is at stake. A 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 would have a single signer and multiple users.[14]

PKE SETUP. The issue 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 ek is a group element whose discrete logarithm is the secret key $dk = \log_G(ek)$, this can be established easily by "hashing into the group" [BF01, BCI+10, WB19]: an agreed-upon public string is hashed to obtain the public key. This is the case for *ElGamal* encryption [ElG85], with which we instantiate PKE in all our implementations.

NIZK SETUP. As with PKE, we can also instantiate NArg with a scheme that has a transparent setup, of which there now exists a host (see the citations on p. 5). We consider *Halo 2* [Com21], *Plonky2* [Zer22] and *Hyperplonk* [CBBZ22] the most promising candidates due to active development efforts and incorporation of the most recent innovations in this rapidly developing field.

A particularly efficient SNARK scheme, with the shortest proofs in the literature so far, is *Groth16* [Gro16]. However, it requires a trusted setup, since it has a "structured" common reference string (CRS). While the setup can be conducted in a distributed manner [BGM17, KMSV21], this still requires trust, so it would be preferable if the signer could set up her own CRS, which would protect her against attacks against soundness. On the other hand, the user relies on NArg being zero-knowledge for blindness, a property that also assumes a CRS that was set up in a trusted way.

The solution to this dilemma are *subversion zero-knowledge* proof systems [BFS16]. This 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. Thus, blindness holds even when the signer sets up the CRS. For *Groth16* Fuchsbauer [Fuc18] defines an algorithm to check that a CRS is well-formed. He shows that, under a "knowledge-type" assumption, if provers only accept well-formed CRSs, then subversion zero knowledge holds. (There are attacks against *Groth16* in which proofs constructed under a malformed CRS leak information on the witness [CGGN17, Fuc19].) Once a CRS is checked (which only needs to be done once), the scheme can be used as usual.

5.2 Hardwiring Parts of the Statement

The efficiency of NArg can be improved by moving elements of the statement θ to the parameters par_{R}. (E.g., multiplication by constants does not require a new gate, as opposed to multiplication

[14] Moreover, the user would have to prove knowledge of the corresponding secret key, so that the unforgeability reduction can extract it.

by variables, in both R1CS [BSCR+18] as well as *Plonk-ish* [GWC19] arithmetization). For relation $\mathsf{R}_{\mathsf{Sch}}$ we can move the signature verification key X and/or the encryption key ek to the relation parameters, which would turn *minimal hardwiring* we considered in Eq. (3), i.e.,

$$\mathsf{R}_{\mathsf{Sch}}((\overbrace{q, \mathbb{G}, G, \mathsf{H}}^{par_{\mathsf{R}}}), (\overbrace{X, R, c, C, prd, ek}^{\theta}), (\overbrace{m, \alpha, \beta, \rho}^{\omega})) \tag{6}$$

into *maximal hardwiring*:

$$\mathsf{R}'_{\mathsf{Sch}}((\overbrace{q, \mathbb{G}, G, \mathsf{H}, X, ek}^{par_{\mathsf{R}'}}), (\overbrace{R, c, C, prd}^{\theta}), (\overbrace{m, \alpha, \beta, \rho}^{\omega})) \ . \tag{7}$$

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. We discuss further implications of this modification in Appendix E.

5.3 Schnorr Parameters

We consider two types of scenarios, which lead to different efficiency of the blind signing protocol:

(A) The group and hash function used for Schnorr $(q, \mathbb{G}, G, \mathsf{H})$ can be chosen in consideration of the specifics of the used **a**rgument system NArg.

(B) The protocol is intended to extend an existing implementation of Schnorr signatures for given parameters $(q, \mathbb{G}, G, \mathsf{H})$, such as the ones used by **B**itcoin.

In scenario (A) we can choose a group that is natively supported by the argument system, as well as an "arithmetic-circuit-friendly" hash function [AGR+16, ACG+19, BGL20, AAB+20, GKR+21]. This means that expressing the computation of H in the language underlying NArg only requires a moderate number of addition and multiplication gates (and/or lookups). This can lead to a CRS of size only a few MBs and proving times of under a second (see Table 1).

Scenario (B) occurs in blockchain protocols when adding (predicate-)blind signing on top of existing Schnorr parameters, which thus have to be handled by the proof system. Standard hash functions like SHA-256 are typically not *arithmetic-circuit-friendly*, which increases the CRS size and proving time.[15]

Second, there might not be implementations of proof systems whose underlying field supports efficient arithmetization of the operation of the group $\mathbb{G}$ used by Schnorr. Non-native simulation of group operations creates significant overhead [SSS+22] and is the main source of inefficiency. This is the case for our implementations using the Bitcoin curve secp256k1 (see Table 1).

5.4 Implementation

To assess the efficiency of our predicate blind signature construction PBSch from Section 4, we benchmark the NArg component, which represents its computational bottleneck. For several

[15] One invocation of the SHA-256 compression function requires around 26 000 R1CS constraints [CGGN17, KPS18, ide]

variants of both scenarios (A) and (B) from Section 5.3 we consider two proof systems: The *Iden3* implementation [ide] of the pre-processing zk-SNARK *Groth16* [Gro16] and the *Fluidex* implementation [Plo] of the universal zk-SNARK *PlonK* [GWC19]. The circuits are written in the domain-specific language of Circom 2.0 and are publicly available [mot].

In Table 1 we give the arithmetic complexity of the relation $\mathsf{R}_{\mathsf{ElGm}}$ from Eq. (8) in terms of number of *constraints* for the considered scenarios.[16] For both proof systems we report the time it takes to compute the witness for the constraint satisfaction problem and the time to compute the proof given this witness. (Witness generation time is equal for both systems, since [Plo] builds on [ide] and reuses their codebase for this.) As the CRS can be split into a "proving key" (used by the user during blind signing) and a "verification key" (used by the signer), we report both sizes. We also state the proof size and the verification time.

Since we discussed the possibility of checking CRS consistency in *Groth16* [Fuc18] to avoid a trusted setup, we also give estimates of its time complexity. In particular, we propose a probabilistic verification of the "proving key" using batching techniques, which we discuss in Appendix F. All experiments were run on an Intel® Core™ i7-10850H CPU @ 2.70GHz × 12 with 31 GB of RAM.

GROUPS AND PKE. We instantiate *Groth16* and *PlonK* over the pairing-friendly curve BN254 [BN06]. The prime order p of the curve group has 254 bits and defines the modulus of the arithmetic circuit over which we instantiate $\mathsf{R}_{\mathsf{Sch}}$; hence all inputs are effectively elements of $\mathbb{F}_p$. We therefore represent the messages in our scheme as elements from $\mathbb{F}_p^n$ for some n (which allows us to handle messages of $n \cdot 253$ bits). This field $\mathbb{F}_p$ is the base field of the curve *Baby JubJub* (BJJ) [BJJ20]. Its elements can be represented as two elements of $\mathbb{F}_p$, and the group operation is efficiently arithmetizable in $\mathbb{F}_p$. To distinguish BJJ elements from the group $\mathbb{G}$ used by Schnorr, *we represent them in* roman *font*, e.g., the generator is G.

We instantiate the encryption scheme PKE (used to encrypt α, β and m in Figure 4) by using the DHIES [ABR98] key encapsulation mechanism of ElGamal [ElG85] over the BJJ curve group and the additive one-time pad over $\mathbb{F}_p$ as the data encapsulation mechanism. That is, to encrypt a message under a public key K, one chooses $\rho \leftarrow\!\!\$\, \mathbb{F}_q$ and computes the hash of ρK. The ciphertext consists of the message blinded by this hash together with the element ρG.

We use an arithmetic-circuit-friendly sponge hash $\Psi_p^{n+2} \colon \mathbb{F}_p^2 \to \mathbb{F}_p^{2+n}$ from the Poseidon family [GKR+21] to obtain field elements $\alpha, \beta, r_1, \ldots, r_n$. The last n elements are used to additively blind the message m (we use α and β directly rather than first choosing and then blinding them). These choices lead to the following instantiation of $\mathsf{R}_{\mathsf{Sch}}$ from Eq. (3) (for which the Schnorr parameters $(q, \mathbb{G}, G, \mathsf{H})$ still depend on the scenario):[17]

[16] Arithmetization is given as a *R1CS relation*, which consists of instance-witness pairs $((A, B, C, \theta), w)$, where A, B, C are matrices and θ, w are vectors over a finite field $\mathbb{F}$, such that $Az \circ Bz = Cz$ for $z := (1, \theta, w)$, where "$\circ$" denotes the entry-wise product [BCR+19]. We refer to each such product as a "constraint" or sometimes "gate".

[17] We stress the importance of type checks when inputs are not $\mathbb{F}_p$ elements, failing which constitutes a common source of error in implementations. Type checks on elements of the statement θ can be directly performed when verifying a NIZK proof, which saves on CRS size and prover time.

$$\mathsf{R}_{\mathsf{ElGm}}((q,\mathbb{G},G,\mathsf{H}),(X,R,c,\mathrm{C},(c_i)_{i\in[n]},prd,\mathrm{K}),((m_i)_{i\in[n]},\rho)) :$$

$$\begin{array}{lll}
(\alpha,\beta,r_1,\dots r_n) \coloneqq \Psi_p^{n+2}(\rho\mathrm{K}) & /\!/ \text{ use Poseidon-DHIES to derive a key} \\
R' \coloneqq R+\alpha G+\beta X & /\!/ \text{ blind } R \text{ in Schnorr group } \mathbb{G} \\
\textbf{return}\ \ c\equiv_q \mathsf{H}(R',X,m_1,\dots,m_n)+\beta & /\!/ \ c \text{ is computed from witness elements} \\
\quad \wedge\ \mathsf{P}(prd,(m_1,\dots,m_n))=1 & /\!/ \ (m_i)_{i\in[n]} \text{ satisfies the predicate } prd \\
\quad \wedge\ \forall i\in[n] : r_i+m_i\equiv_p c_i & /\!/ \text{ check consistency of DHIES } \dots \\
\quad \wedge\ \mathrm{C}=\rho\mathrm{G} & /\!/ \ \dots \text{encryption in the } \mathsf{BJB} \text{ group}
\end{array} \tag{8}$$

This instantiation of PKE leads to a small circuit size of $\mathsf{R}_{\mathsf{ElGm}}$, since "sponge squeezing" for Ψ_p has very low complexity compared to standard ElGamal encryption of the individual components (due to the required variable-base group multiplications; it would also require cumbersome mappings of messages to group elements). The outputs of Ψ_p are in $\mathbb{F}_p$. Since $p=(8-\epsilon)\cdot q$ with $\epsilon<2^{-124}$, taking uniform values in $\mathbb{F}_p$ modulo q is statistically close to uniform values in $\mathbb{F}_q$. Thus, assuming security of DHIES with Poseidon, α and β modulo q are distributed (almost) as required.

Note that the first line of the return statement in (8) checks a congruence modulo q, whereas the third line is modulo p. The former operation is typically not "natively" supported by the proof system, but its overhead can be kept low. Specifically, when Schnorr is implemented over the BJB curve, the check $c\equiv_q \mathsf{H}(R',X,m_1,\dots,m_n)+\beta$ can be done via $c\mathrm{G}=\mathsf{H}(R',X,m_1,\dots,m_n)\mathrm{G}+\beta\mathrm{G}$, where the verifier computes $c\mathrm{G}$. If X is hard-wired (cf. Section 5.2) then $c\mathrm{X}=\mathsf{H}(R',X,m_1,\dots,m_n)\mathrm{X}+\beta\mathrm{X}$ is even cheaper, since $\beta\mathrm{X}$ is already computed during the blinding process of R. Alternatively, if q has "special form"[18] [CP05, §9.2.3], then the one-time modulo reduction can be performed directly at a cost even lower compared to the above technique for fixed X.

Optimized Schnorr parameters. We start with scenario (A) (see Section 5.3) considering NArg-"friendly" choices of the group $\mathbb{G}$ and the hash function H. We choose the same BJB group and Poseidon hash function as in the implementation of PKE. Performance details of an implementation of fully blind signatures in this configuration for 253-byte messages are given in Table 1, column (A2).

A *run-time optimized and minimalist* scenario is (A1), where we **hardwire** the signature verification key X and the encryption key ek (as in Eq. (7)) and support 253-*bit* messages.[19]

In scenario (A3), we implement **partially** blind signatures for messages that have 126 public bytes and 126 secret bytes. We optimized the generic construction from any PBS for the predicate from Eq. (1) as follows. Since only messages $m=(info,m')$ with $info=prd$ will be signed, it suffices if the user encrypts m' instead of m in its first protocol message. This

[18] *Special form* refers to a modulus of the form $N=2^a+b$ for positive a and "small" b. This is unfortunately not the case for the group orders of secp256k1 and BJB, for which the bitlength of b is 129 and 249 respectively. The algorithm however performs relatively well for the base field size of Curve25519 and secp256k1.

[19] Since scalar multiplication ("exponentiation", using multiplicative notation) in the BJB group for *fixed* base requires roughly 770 R1CS constraints as opposed about 2 530 constraints for variable base (incl. bit-decomposition; in the current [ide] implementation), the bulk of constraints saved from (A2) to (A1) comes from this hardwiring aspect, rather than the smaller message size.

Table 1. Benchmark NArg for the relation $\mathsf{R}_{\mathsf{ElGm}}$ in Eq. (8) for different scenarios. The first two rows specify the used Schnorr parameters. 'Hardwiring' can be maximal Eq. (7) or minimal Eq. (6). 'Constraints' capture the complexity of the arithmetization of $\mathsf{R}_{\mathsf{ElGm}}$. The time it takes to generate a proof is the sum of 'witness generation' and 'proving time'. We present proving key sizes after applying point compression. Proving and proof-verification times were measured using the system command `time` when run on an Intel® Core™ i7-10850H CPU @ 2.70GHz×12 with 31 GB of RAM. (Note that results depend heavily on machine specifics; e.g., in [Bot] **Groth16** proof verification is reported to be 100 times faster than our numbers using the same zkSNARK library.) '*pk* verif. time' is an estimate on proving-key verification based on results discussed in Appendix F.

Scenario	**(A1)**	**(A2)**	**(A3)**	**(B1)**	**(B2)**	**(B3)**
Schnorr curve	BJB	BJB	BJB	secp256k1	secp256k1	BJB
Schnorr hash	Poseidon	Poseidon	Poseidon	SHA-256	SHA-256	SHA-256
Blindness type	full	full	partial	full	predicate	predicate
Message size	253 b	253 B	252 B	256 b	256 b	256 b
Hardwiring	max.	min.	min.	min.	min.	min.
Constraints	3 353	7 957	7 531	1 564 556	1 716 794	218 867
Witness generation	13 ms	14 ms	14 ms	23 s	24 s	0.1 s
Proving system	**Groth16** [Gro16] implementation of [ide]					
Prov. key (*pk*) size	0.8 MB	2.15 MB	2.05 MB	550 MB	550 MB	62.5 MB
pk verif. time (≈)	0.64 s	0.97 s	0.93 s	3h 55min	4h 43min	4 min 38 s
Verif. key size	1.75 kB	2.75 kB	2.1 kB	3.75 kB	3.4 kB	2.2 kB
Statement size	328 B	1.2 kB	900 kB	979 B	1.2 kB	1.2 kB
Proving time	0.5 s	0.7 s	0.7 s	37 s	39 s	4.7 s
Proof size	402 B					
Proof verif. time	0.4 s					
Proving system	**Plonk** [GWC19] implementation of [Plo]					
Proving key size	0.92 MB	1.6 MB	1.5 MB	336 MB	354 MB	60.5 MB
Verif. key size	0.55 kB	0.55 kB	0.55 kB	2.75 kB	0.55 kB	0.55 kB
Statement size	289 B	1.4 kB	791 B	553 B	562 B	585 B
Proving time	1.2 s	1.5 s	1.7 s	2 m 27 s	2 m 29 s	33.482 s
Proof size	0.6 kB	0.8 kB	0.75 kB	0.9 kB	0.9 kB	0.7 kB
Verification time	12 ms					

makes partially blind signing slightly more performant than fully blind signing for messages of the same length, since only m' is contained in the witness of the circuit.

Fixed Schnorr parameters. As a concrete scenario of type (B), we consider blind signing of **Bitcoin transactions**, that is, blind issuing of Schnorr signatures (supported by Bitcoin since the *Taproot* upgrade [WNR20]) over the group secp256k1 and using SHA-256. We consider "Pay To Public Key Hash", which is the most common form of pubkey script when creating a transaction. A serialized transaction is hashed twice using SHA-256 and then signed [Wik]; we thus need to handle 256-bit messages.

Scenario (B1) in Table 1 gives performance upper bounds for fully blind signature issuing, which deteriorate compared to scenario (A). The main reason is that secp256k1 and SHA-256 are not efficiently arithmetizable in existing SNARK implementations. Moreover, we used the (currently only available) prototype implementation of the secp256k1 curve by [0xP] (written in the Circom language), which is not optimized.

Scenario (B2) now leverages the possibilities offered by **predicate** blind signatures. We consider a signer that blindly signs a transaction, *but wants to ensure that the transferred amount is below a certain threshold*. Since Bitcoin signs the hash of a transaction, this requires PBS for NP-relations (Section 4.3). Concretely, the relation $\mathsf{P}(prd, m, w)$ takes as witness w the transaction (254 bytes suffice for a standard Bitcoin transaction), checks if the transaction value is smaller than a value specified by prd and whether the hash of w equals m.

Scenario (B3) has the same functionality, but uses the curve BJB instead of secp256k1. This could give a very rough estimate for what performance could be achieved when using a NIZK for which the curve secp256k1 has an efficient arithmetization. We discuss avenues for constructing such a NIZK in Appendix G.

Acknowledgements. This work has been funded by the Vienna Science and Technology Fund (WWTF) [10.47379/VRG18002]. We would like to thank Tim Ruffing for preliminary discussions.

References

[0xP] 0xPARC. Big integer arithmetic and secp256k1 ecc operations in circom. Available at https://github.com/0xPARC/circom-ecdsa.

[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.

[ABR98] Michel Abdalla, Mihir Bellare, and Phillip Rogaway. DHIES: An encryption scheme based on the Diffie-Hellman problem. Contributions to IEEE P1363a, September 1998.

[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*, pp. 371–397. Springer, December 2019.

[AFG+10] Masayuki Abe, Georg Fuchsbauer, Jens Groth, Kristiyan Haralambiev, and Miyako Ohkubo. Structure-preserving signatures and commitments to group elements. In Tal Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pp. 209–236. Springer, August 2010.

[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*, pp. 191–219. Springer, December 2016.

[AO00] Masayuki Abe and Tatsuaki Okamoto. Provably secure partially blind signatures. In Mihir Bellare, editor, *CRYPTO 2000*, volume 1880 of *LNCS*, pp. 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*, pp. 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*, pp. 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*, pp. 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*, pp. 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*, pp. 237–254. Springer, August 2010.

[BCK+22] Mihir Bellare, Elizabeth C. Crites, Chelsea Komlo, Mary Maller, Stefano Tessaro, and Chenzhi Zhu. Better than advertised security for non-interactive threshold signatures. In *CRYPTO 2022, Part IV*, LNCS, pp. 517–550. Springer, August 2022.

[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*, pp. 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*, pp. 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*, pp. 103–128. Springer, May 2019.

[BCTV14] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Scalable zero knowledge via cycles of elliptic curves. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part II*, volume 8617 of *LNCS*, pp. 276–294. Springer, August 2014.

[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.

[BDLO12] Daniel J. Bernstein, Jeroen Doumen, Tanja Lange, and Jan-Jaap Oosterwijk. Faster batch forgery identification. In Steven D. Galbraith and Mridul Nandi, editors, *INDOCRYPT 2012*, volume 7668 of *LNCS*, pp. 454–473. Springer, December 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*, pp. 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*, pp. 213–229. Springer, August 2001.

[BFI+10] Olivier Blazy, Georg Fuchsbauer, Malika Izabachène, Amandine Jambert, Hervé Sibert, and Damien Vergnaud. Batch Groth-Sahai. In Jianying Zhou and Moti Yung, editors, *ACNS 10*, volume 6123 of *LNCS*, pp. 218–235. Springer, June 2010.

[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*, pp. 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*, pp. 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*, pp. 587–617. Springer, December 2021.

[BFQ21] Balthazar Bauer, Georg Fuchsbauer, and Chen Qian. Transferable E-cash: A cleaner model and the first practical instantiation. In Juan Garay, editor, *PKC 2021, Part II*, volume 12711 of *LNCS*, pp. 559–590. Springer, May 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*, pp. 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.

[BJB20] WhiteHat Barry, Baylina Jordi, and Marta Bellés. Baby jubjub elliptic curve. *Ethereum Improvement Proposal, EIP-2494*, 29, 2020.

[BK22] Dan Boneh and Chelsea Komlo. Threshold signatures with private accountability. In *CRYPTO 2022, Part IV*, LNCS, pp. 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*, pp. 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*, pp. 33–53. Springer, October 2021.

[BLS01] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. In Colin Boyd, editor, *ASIACRYPT 2001*, volume 2248 of *LNCS*, pp. 514–532. Springer, December 2001.

[BN06] Paulo S. L. M. Barreto and Michael Naehrig. Pairing-friendly elliptic curves of prime order. In Bart Preneel and Stafford Tavares, editors, *SAC 2005*, volume 3897 of *LNCS*, pp. 319–331. Springer, August 2006.

[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*, pp. 31–46. Springer, January 2003.

[Bot] Gautam Botrel. gnark: high-performance, open-source library that enables effective zksnark applications. Available at https://consensys.net/blog/research-development/gnark-your-guide-to-write-zksnarks-in-go/.

[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*, pp. 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*, pp. 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.

[BSCR+18] Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P. Ward. Aurora: Transparent succinct arguments for R1CS. Cryptology ePrint Archive, Report 2018/828, 2018. https://eprint.iacr.org/2018/828.

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

[CHL+22] Rutchathon Chairattana-Apirom, Lucjan Hanzlik, Julian Loss, Anna Lysyanskaya, and Benedikt Wagner. PI-cut-choo and friends: Compact blind signatures via parallel instance cut-and-choose and more. In *CRYPTO 2022, Part III*, LNCS, pp. 3–31. Springer, August 2022.

[CBBZ22] Binyi Chen, Benedikt Bünz, Dan Boneh, and Zhenfei Zhang. HyperPlonk: Plonk with linear-time prover and high-degree custom gates. Cryptology ePrint Archive, Report 2022/1355, 2022. https://eprint.iacr.org/2022/1355.

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

[CG08] Jan Camenisch and Thomas Groß. Efficient attributes for anonymous credentials. In Peng Ning, Paul F. Syverson, and Somesh Jha, editors, *ACM CCS 2008*, pp. 345–356. ACM Press, October 2008.

[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*, pp. 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*, pp. 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*, pp. 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*, pp. 738–768. Springer, May 2020.

[CKM21] Elizabeth Crites, Chelsea Komlo, and Mary Maller. How to prove Schnorr assuming Schnorr: Security of multi- and threshold signatures. Cryptology ePrint Archive, Paper 2021/1375, 2021. https://eprint.iacr.org/2021/1375.

[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*, pp. 93–118. Springer, May 2001.

[CL04] Jan Camenisch and Anna Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In Matthew Franklin, editor, *CRYPTO 2004*, volume 3152 of *LNCS*, pp. 56–72. Springer, August 2004.

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

[Cor00] Jean-Sébastien Coron. On the exact security of full domain hash. In Mihir Bellare, editor, *CRYPTO 2000*, volume 1880 of *LNCS*, pp. 229–235. Springer, August 2000.

[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*, pp. 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*, pp. 89–105. Springer, August 1993.

[CP05] Richard E Crandall and Carl Pomerance. *Prime numbers: a computational perspective*, volume 2. Springer, 2005.

[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.

[DN07] Cynthia Dwork and Moni Naor. Zaps and their applications. *SIAM Journal on Computing*, 36(6):1513–1543, 2007.

[EL] Electron-Labs. Ed25519 implementation in circom. Available at https://github.com/Electron-Labs/ed25519-circom.

[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.

[FGHP09] Anna Lisa Ferrara, Matthew Green, Susan Hohenberger, and Michael Østergaard Pedersen. Practical short signature batch verification. In Marc Fischlin, editor, *CT-RSA 2009*, volume 5473 of *LNCS*, pp. 309–324. Springer, April 2009.

[FHKS16] Georg Fuchsbauer, Christian Hanser, Chethan Kamath, and Daniel Slamanig. Practical round-optimal blind signatures in the standard model from weaker assumptions. In Vassilis Zikas and Roberto De Prisco, editors, *SCN 16*, volume 9841 of *LNCS*, pp. 391–408. Springer, August/September 2016.

[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*, pp. 233–253. Springer, August 2015.

[Fis06] Marc Fischlin. Round-optimal composable blind signatures in the common reference string model. In Cynthia Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pp. 60–77. Springer, August 2006.

[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*, pp. 33–62. Springer, August 2018.

[FKP16] Manuel Fersch, Eike Kiltz, and Bertram Poettering. On the provable security of (EC)DSA signatures. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pp. 1651–1662. ACM Press, October 2016.

[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*, pp. 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*, pp. 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*, pp. 657–689. Springer, May 2019.

[FP09] Georg Fuchsbauer and David Pointcheval. Proofs on encrypted values in bilinear groups and an application to anonymity of signatures. In Hovav Shacham and Brent Waters, editors, *PAIRING 2009*, volume 5671 of *LNCS*, pp. 132–149. Springer, August 2009.

[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*, pp. 63–95. Springer, May 2020.

[FST10] David Freeman, Michael Scott, and Edlyn Teske. A taxonomy of pairing-friendly elliptic curves. *Journal of Cryptology*, 23(2):224–280, April 2010.

[Fuc11] Georg Fuchsbauer. Commuting signatures and verifiable encryption. In Kenneth G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pp. 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*, pp. 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*, pp. 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*, pp. 477–495. Springer, May 2014.

[Gha17] Essam Ghadafi. Efficient round-optimal blind signatures in the standard model. In Aggelos Kiayias, editor, *FC 2017*, volume 10322 of *LNCS*, pp. 455–473. Springer, April 2017.

[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*, pp. 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*, pp. 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.

[Gro10] Jens Groth. Short pairing-based non-interactive zero-knowledge arguments. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pp. 321–340. Springer, December 2010.

[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*, pp. 305–326. Springer, May 2016.

[GRS+11] Sanjam Garg, Vanishree Rao, Amit Sahai, Dominique Schröder, and Dominique Unruh. Round optimal blind signatures. In Phillip Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pp. 630–648. Springer, August 2011.

[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.

[HBG16] Ethan Heilman, Foteini Baldimtsi, and Sharon Goldberg. Blindly signed contracts: Anonymous on-blockchain and off-blockchain bitcoin transactions. In Jeremy Clark, Sarah Meiklejohn, Peter Y. A. Ryan, Dan S. Wallach, Michael Brenner, and Kurt Rohloff, editors, *FC 2016 Workshops*, volume 9604 of *LNCS*, pp. 43–60. Springer, February 2016.

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

[HIP+] Scott Hendrickson, Jana Iyengar, Tommy Pauly, Steven Valdez, and Christopher A. Wood. Private Access Tokens. Internet-Draft draft-private-access-tokens-00, Internet Engineering Task Force. Work in Progress.

[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.

[HKKL07] Carmit Hazay, Jonathan Katz, Chiu-Yuen Koo, and Yehuda Lindell. Concurrently-secure blind signatures without random oracles or setup assumptions. In Salil P. Vadhan, editor, *TCC 2007*, volume 4392 of *LNCS*, pp. 323–341. Springer, February 2007.

[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*, pp. 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*, pp. 236–250. Springer, February/March 2006.

[HLW22] Lucjan Hanzlik, Julian Loss, and Benedikt Wagner. Rai-choo! evolving blind signatures to the next level. Cryptology ePrint Archive, Paper 2022/1350, 2022. https://eprint.iacr.org/2022/1350.

[Hou21] Youssef El Housni. Benchmarking pairing-friendly elliptic curves libraries, 2021. Available at https://hackmd.io/@gnark/eccbench.

[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.

[ide] iden3. Circom 2.0. Available at https://iden3.io/circom.

[JLE17] Bargav Jayaraman, Hannah Li, and David Evans. Decentralized certificate authorities. *CoRR*, abs/1706.03370, 2017.

[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*, pp. 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.

[KLR21] Jonathan Katz, Julian Loss, and Michael Rosenberg. Boosting the security of blind signature schemes. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part IV*, volume 13093 of *LNCS*, pp. 468–492. Springer, December 2021.

[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*, pp. 468–497. Springer, 2022.

[KMSV21] Markulf Kohlweiss, Mary Maller, Janno Siim, and Mikhail Volkhov. Snarky ceremonies. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part III*, volume 13092 of *LNCS*, pp. 98–127. Springer, December 2021.

[KPS18] Ahmed E. Kosba, Charalampos Papamanthou, and Elaine Shi. xJsnark: A framework for efficient verifiable computation. In *2018 IEEE Symposium on Security and Privacy*, pp. 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.

[KZ06] Aggelos Kiayias and Hong-Sheng Zhou. Concurrent blind signatures without random oracles. In Roberto De Prisco and Moti Yung, editors, *SCN 06*, volume 4116 of *LNCS*, pp. 49–62. Springer, September 2006.

[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*, pp. 339–346. Springer, October 2019.

[mot] mottla. (Concurrently secure) blind Schnorr signature reference circuits. Available at https://github.com/mottla/Blind-Schnorr-Signatures.

[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*, pp. 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*, pp. 80–99. Springer, March 2006.

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

[Pip76] Nicholas Pippenger. On the evaluation of powers and related problems. In *17th Annual Symposium on Foundations of Computer Science (sfcs 1976)*, pp. 258–263. IEEE Computer Society, 1976.

[Plo] Plonkit. A zksnark toolkit to work with circom zkp dsl in plonk proof system. Available at https://github.com/fluidex/plonkit.

[Poi98] David Pointcheval. Strengthened security for blind signatures. In Kaisa Nyberg, editor, *EUROCRYPT'98*, volume 1403 of *LNCS*, pp. 391–405. Springer, May/June 1998.

[PS96] David Pointcheval and Jacques Stern. Security proofs for signature schemes. In Ueli M. Maurer, editor, *EUROCRYPT'96*, volume 1070 of *LNCS*, pp. 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.

[Rad68] Charles M Rader. Discrete fourier transforms when the number of data samples is prime. *Proceedings of the IEEE*, 56(6):1107–1108, 1968.

[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*, pp. 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*, pp. 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*, pp. 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*, pp. 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.

[SSS+22] Huachuang Sun, Haifeng Sun, Kevin Singh, Akhil Sai Peddireddy, Harshad Patil, Jianwei Liu, and Weikeng Chen. The inspection model for zero-knowledge proofs and efficient Zerocash with secp256k1 keys. 2022. Available at https://eprint.iacr.org/2022/1079.

[TZ22] Stefano Tessaro and Chenzhi Zhu. Short pairing-free blind signatures with exponential security. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part II*, volume 13276 of *LNCS*, pp. 782–811. Springer, May/June 2022.

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

[WB19] Riad S. Wahby and Dan Boneh. Fast and simple constant-time hashing to the BLS12-381 elliptic curve. *IACR TCHES*, 2019(4):154–179, 2019.

[Wik] Bitcoin Wiki. The op_checksig script opcode. Available at https://en.bitcoin.it/wiki/OP_CHECKSIG.

[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.

[Yao82] Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *23rd FOCS*, pp. 160–164. IEEE Computer Society Press, November 1982.

[Zer22] Polygon Zero. Plonky2: Fast recursive arguments with PLONK and FRI, 2022. Available at https://github.com/mir-protocol/plonky2.

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 any of the group elements obtained from a challenge oracle, while being given access to a discrete-logarithm oracle that can be called on all other elements. In contrast to the original OMDL game [BNPS03], here the DL oracle can *only be queried on challenge group elements*, as opposed to arbitrary group elements. This makes the wOMDL assumption significantly weaker than OMDL; in particular, it is implied by DL (while such an implication is unlikely to hold for OMDL [BFL20]). The reduction embeds its DL challenge randomly in one of the wOMDL adversary's challenges, which results in a security loss linear in the number of challenge queries. Using a proof technique by Coron [Cor00], we reduce the loss to the number of DL oracle calls.

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

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

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

$\underline{\mathsf{wOMDL}^{\mathsf{A}}_{\mathsf{GrGen}}(1^\lambda)}$
$(q, \mathbb{G}, G) \leftarrow \mathsf{GrGen}(1^\lambda)$
$\vec{x} := [\,]\,;\ \vec{q} := [\,]$
$y \leftarrow \mathsf{A}^{\textsc{Chal},\textsc{DLog}}(q, \mathbb{G}, G)$
return $\left(|\vec{x}| > 0 \wedge y \in \vec{x} \wedge y \notin \vec{q}\right)$

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

$\underline{\textsc{DLog}(i)}$
$\vec{q} = \vec{q} \| \vec{x}_i$
return $\vec{x}_i$

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

$$\mathsf{Adv}^{\mathsf{wOMDL}}_{\mathsf{GrGen},\mathsf{A}}(\lambda) \;\leq\; q \frac{1}{\left(1 - \frac{1}{q+1}\right)^{q+1}} \cdot \mathsf{Adv}^{\mathsf{DL}}_{\mathsf{GrGen},\mathsf{B}}(\lambda)\ , \tag{9}$$

which for large q *approaches*

$$\mathsf{Adv}^{\mathsf{wOMDL}}_{\mathsf{GrGen},\mathsf{A}}(\lambda) \;\simeq\; q \cdot \exp(1) \cdot \mathsf{Adv}^{\mathsf{DL}}_{\mathsf{GrGen},\mathsf{B}}(\lambda)\ . \tag{10}$$

Proof. We construct B playing against DL, which on input $(q, \mathbb{G}, G, Z)$ must compute $\log_G(Z)$. B simulates game wOMDL for A, except that, when answering a call to CHAL(), with probability $1 - P$, for some value P, it embeds Z in its response and aborts if A ever queries DLOG at a position at which Z was embedded:

$\mathsf{B}(q, \mathbb{G}, G, Z)$	CHAL()	DLOG(i)
$\vec{x} \coloneqq [\,]$ // empty list of tuples	$\delta \leftarrow\!\!\$\, [0, 1]$	**if** $\vec{x}_i[0] = 0$:
$y \leftarrow \mathsf{A}^{\text{CHAL,DLOG}}(q, \mathbb{G}, G)$	$x \leftarrow\!\!\$\, \mathbb{Z}_q$; $X \coloneqq xG$	**abort**
foreach (b, x) **in** $\vec{x}$:	**if** $\delta > P$: $\vec{x} = \vec{x} \| (0, x)$	**return** $\vec{x}_i[1]$
if $Z = (y - x)G$:	**return** $Z + X$	
return $y - x$	$\vec{x} = \vec{x} \| (1, x)$	
return 0	**return** X	

Consider an adversary A against wOMDL that makes q DLOG queries. The probability that B does not abort its simulation is at least P^q. If B does not abort, the simulation is perfect. Moreover, if A wins wOMDL, then the probability that its output y corresponds to an entry $(0, x_i)$ is $1 - P$. In this case $y = \log(Z + x_i G)$ and thus B returns $\log Z$. Since we must have $y \notin \vec{q}$, this probability is independent of B's abort probability and B succeeds thus with probability at least $\alpha(P) \coloneqq P^q \cdot (1 - P)$. Since $\alpha(P)$ is maximal for $P_{max} = \frac{q}{q+1}$, we obtain $\alpha(P_{max}) = \frac{1}{q}(1 - \frac{1}{q+1})^{q+1}$. □

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.

$\underline{\mathsf{G}_0}$. This is game UNF from Figure 2 with PBS instantiated by PBSch from Figure 4. The generic PBS.Setup hence is replaced by the setup from Figure 4. In SIGN_1, the call $\mathsf{PBS.Sign}_1$ is instantiated by sampling $r \leftarrow\!\!\$\, \mathbb{Z}_q$ and returning $R = rG$, and in 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 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 SIGN_2 call on input $(j, (c, \pi))$, we check if for the decrypted values at index j, we either have $c \not\equiv_q \mathsf{H}(R', X, \vec{\mathrm{M}}_j) + \vec{\beta}_j$ 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.

$\underline{\text{REDUCTION FROM SOUNDNESS OF NArg.}}$ We show that the difference between $\mathsf{Adv}^{\mathsf{UNF}}_{\mathsf{PBSch,A}}(\lambda)$ and $\mathsf{Adv}^{\mathsf{G}_1}_{\mathsf{A}}(\lambda)$ is bounded by the advantage in winning the game SND (Definition 4) against soundness of $\mathsf{NArg}[\mathsf{R}_{\mathsf{Sch}}]$ played by adversary S which returns the statement/proof pair whenever G_1 aborts (defined in Figure 6).

According to the definition of game SND for NArg for relation $\mathsf{R}_{\mathsf{Sch}}$, S gets as input the common reference string crs, generated by $\mathsf{NArg.Setup}(par_{\mathsf{R}})$, where par_{R} is generated by NArg.Rel, which is defined as Sch.Setup. Reduction S, run in game SND therefore perfectly simulates G_1 to adversary A until abort. In SIGN_2, it checks whether for a valid statement/proof

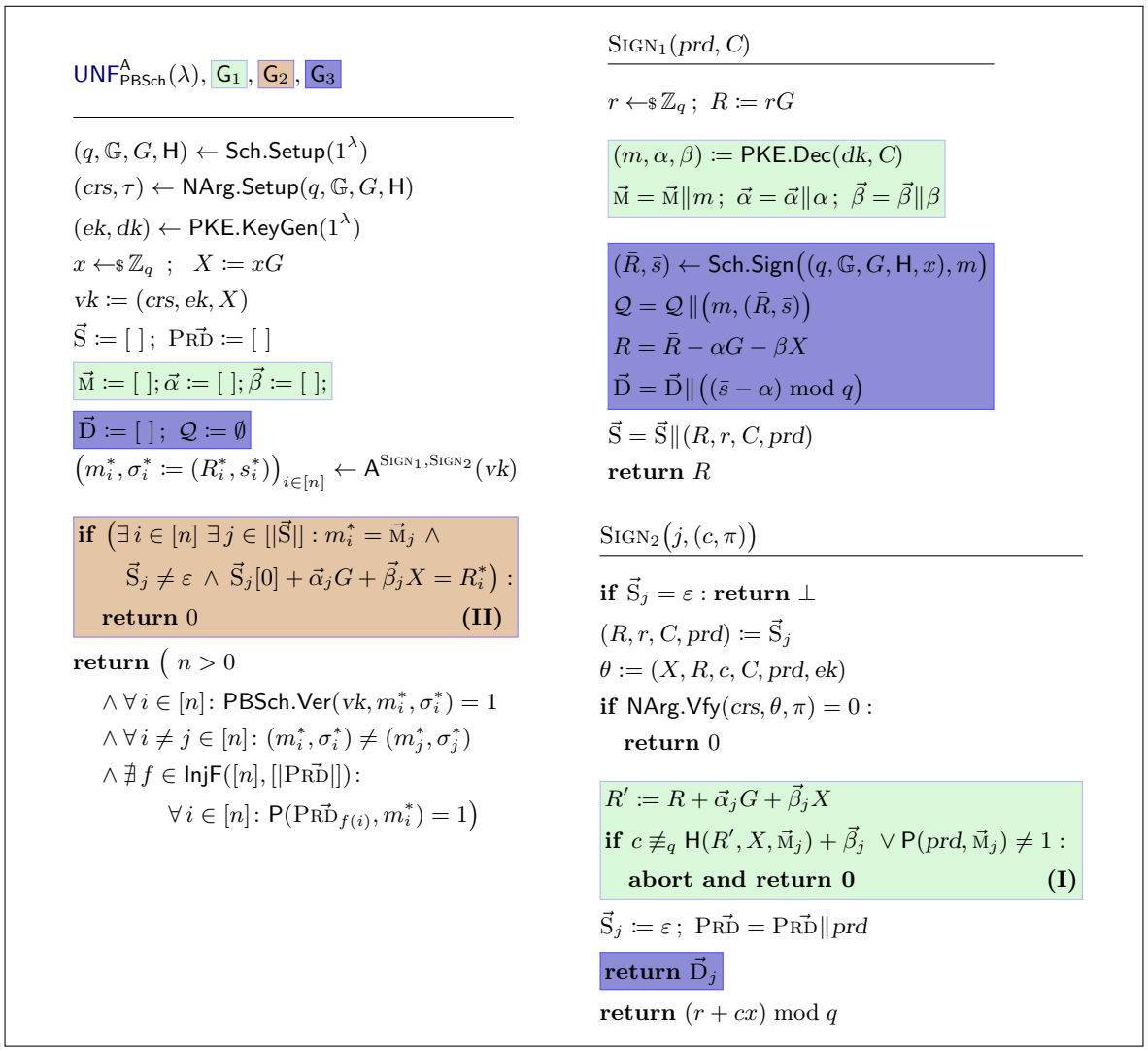

Fig. 5. The unforgeability game from Figure 2 for the scheme $\mathsf{PBSch}[\mathsf{P}, \mathsf{GrGen}, \mathsf{HGen}, \mathsf{PKE}, \mathsf{NArg}]$ from Figure 4 and hybrid games used in the proof of Theorem 1. G_i includes all boxes with an index $\leq i$ and ignores all boxes with and index $> i$.

pair (θ, π) parts of the supposed witness m, α and β satisfy $c \neq (\mathsf{H}(R', X, m) + \beta) \bmod q$ 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 line (I) in a call SIGN_2 on input $(j, (c, \pi))$ and let $(\theta := (X, R, c, C, prd, ek), \pi)$ 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. $\mathsf{R}_{\mathsf{Sch}}(par_{\mathsf{R}}, \theta, w') = 1$, which means:

$$c \equiv_q \mathsf{H}(R', X, m') + \beta' \ \wedge\ \mathsf{P}(prd, m') = 1 \ \wedge\ \mathsf{PKE.Enc}(ek, (m', \alpha', \beta'); \rho') = C\,, \tag{11}$$

$\mathsf{S}(crs)$

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

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

$r \leftarrow\!\!\$\, \mathbb{Z}_q \,;\; R := rG$

$(m, \alpha, \beta) := \mathsf{PKE.Dec}(dk, C)$
$\vec{\mathrm{M}} = \vec{\mathrm{M}} \| m \,;\; \vec{\alpha} = \vec{\alpha} \| \alpha \,;\; \vec{\beta} = \vec{\beta} \| \beta$

$\vec{\mathrm{S}} = \vec{\mathrm{S}} \| (R, r, C, prd)$
return R

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

if $\vec{\mathrm{S}}_j = \varepsilon$: **return** $\bot$
$(R, r, C, prd) := \vec{\mathrm{S}}_j$
$\theta := (X, R, c, C, prd, ek)$
if $\mathsf{NArg.Vfy}(par, \theta, \pi) = 0$:
  return 0

$R' := R + \vec{\alpha}_j G + \vec{\beta}_j X$
if $c \not\equiv_q \mathsf{H}(R', X, \vec{\mathrm{M}}_j) + \vec{\beta}_j \;\vee \mathsf{P}(prd, \vec{\mathrm{M}}_j) \neq 1$:
  stop and return (θ, π)

$\vec{\mathrm{S}}_j := \varepsilon$
return $\big((r + cx) \bmod q\big)$

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

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. (11), this can only be the case if

$$m' = m \quad \text{and} \quad \alpha' = \alpha \quad \text{and} \quad \beta' = \beta \;. \tag{12}$$

Again by the definition of S, we have $c \not\equiv_q \mathsf{H}(R + \alpha G + \beta X, X, m) + \beta \;\vee\; \mathsf{P}(prd, m) \neq 1$, which is a contradiction to Eq. (11) and (12). Therefore such a witness w' does not exist. This means that whenever (I) is reached in G_1, then S wins SND and thus

$$\mathsf{Adv}^{\mathsf{UNF}}_{\mathsf{PBSch},\mathsf{A}}(\lambda) \leq \mathsf{Adv}^{\mathsf{SND}}_{\mathsf{NArg}[\mathsf{R}_{\mathsf{Sch}}],\mathsf{S}}(\lambda) + \Pr\big[\mathsf{G}_1^{\mathsf{A}}(\lambda)\big] \;. \tag{13}$$

$\underline{\mathsf{G}_2}$. In G_2 we introduce the event

$$\mathsf{E} \;:\Leftrightarrow\; \exists\, i \in [n]\; \exists\, j \in [|\vec{\mathrm{S}}|] : m_i^* = \vec{\mathrm{M}}_j \;\wedge\; \vec{\mathrm{S}}_j \neq \varepsilon \;\wedge\; \vec{\mathrm{S}}_j[0] + \vec{\alpha}_j G + \vec{\beta}_j X = R_i^* \;, \tag{14}$$

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 $\Pr\big[\mathsf{G}_2^{\mathsf{A}}(\lambda)\big] = \Pr\big[\mathsf{G}_1^{\mathsf{A}}(\lambda) \wedge \neg\mathsf{E}\big]$. Together with $\Pr\big[\mathsf{G}_1^{\mathsf{A}}(\lambda)\big] = \Pr\big[\mathsf{G}_1^{\mathsf{A}}(\lambda) \wedge \mathsf{E}\big] + \Pr\big[\mathsf{G}_1^{\mathsf{A}}(\lambda) \wedge \neg\mathsf{E}\big]$ we obtain:

$$\Pr\big[\mathsf{G}_1^{\mathsf{A}}(\lambda)\big] = \Pr\big[\mathsf{G}_1^{\mathsf{A}}(\lambda) \wedge \mathsf{E}\big] + \Pr\big[\mathsf{G}_2^{\mathsf{A}}(\lambda)\big] \;. \tag{15}$$

REDUCTION TO DL. We bound $\Pr[\mathsf{G}_1^{\mathsf{A}}(\lambda) \wedge \mathsf{E}]$ 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.

$\mathsf{L}^{\text{CHAL},\text{DLOG}}(q, \mathbb{G}, G)$

$\mathsf{H} \leftarrow \mathsf{HGen}(q)$
$(crs, \tau) \leftarrow \mathsf{NArg.Setup}\big((q, \mathbb{G}, G, \mathsf{H})\big)$
$(ek, dk) \leftarrow \mathsf{PKE.KeyGen}(1^\lambda)$
$x \leftarrow\!\!\$\, \mathbb{Z}_q \;\; ; \;\; X := xG$
$vk := (crs, ek, X)$
$\vec{\mathrm{S}} := [\,]\,; \;\vec{\mathrm{PRD}} := [\,]$
$\vec{\mathrm{M}} := [\,]; \vec{\alpha} := [\,]; \vec{\beta} := [\,]$
$\big(m_i^*, (R_i^*, s_i^*)\big)_{i\in[n]} \leftarrow \mathsf{A}^{\text{SIGN}_1,\text{SIGN}_2}(vk)$

if $\big(\exists\, i \in [n]\ \exists\, j \in [|\vec{\mathrm{S}}|] : m_i^* = \vec{\mathrm{M}}_j \wedge$
$\vec{\mathrm{S}}_j \neq \varepsilon \wedge \vec{\mathrm{S}}_j[0] + \vec{\alpha}_j G + \vec{\beta}_j X = R_i^*\big)$:
$r_j := \big(s_i^* - \vec{\alpha}_j - \vec{\beta}_j \cdot x$
$\quad - \mathsf{H}(R_i^*, X, m_i^*) \cdot x\big) \bmod q$
return r_j
return $\bot$

$\text{SIGN}_1(prd, C)$

$R \leftarrow \text{CHAL}()$
$(m, \alpha, \beta) := \mathsf{PKE.Dec}(dk, C)$
$\vec{\mathrm{M}} = \vec{\mathrm{M}} \| m \,; \;\vec{\alpha} = \vec{\alpha} \| \alpha \,; \;\vec{\beta} = \vec{\beta} \| \beta$
$\vec{\mathrm{S}} = \vec{\mathrm{S}} \| (R, C, prd)$
$R' := R + \alpha G + \beta X$
return R

$\text{SIGN}_2\big(j, (c, \pi)\big)$

if $\vec{\mathrm{S}}_j = \varepsilon$: **return** $\bot$
$(R, C, prd) := \vec{\mathrm{S}}_j$
$\theta := (X, R, c, C, prd, ek)$
if $\mathsf{NArg.Vfy}(crs, \theta, \pi) = 0$:
return $\bot$

$R' := R + \vec{\alpha}_j G + \vec{\beta}_j X$
if $c \not\equiv_q \mathsf{H}(R', X, \vec{\mathrm{M}}_j) + \vec{\beta}_j \;\vee \mathsf{P}(prd, \vec{\mathrm{M}}_j) \neq 1$:
abort and return 0 **(I)**
$\vec{\mathrm{S}}_j := \varepsilon; \vec{\mathrm{PRD}} = \vec{\mathrm{PRD}} \| prd$
$r := \text{DLOG}(j)$
return $\big((r + cx) \bmod q\big)$

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

By the definition of game wOMDL, L receives as input the group parameters $(q, \mathbb{G}, G)$. With that it chooses a hash function $\mathsf{H} \leftarrow \mathsf{HGen}(q)$ and then simulates G_1 for A, where in each call of SIGN_1, L queries its challenge oracle $R \leftarrow \text{CHAL}()$. Since the oracle returns uniformly sampled elements, the simulation is perfect up to this point. If A closes a session with session number j successfully with a call to SIGN_2, L obtains $r := \text{DLOG}(j)$ from its oracle DLOG.

Assume A satisfies condition E from (14). Thus some session number j with challenge $R_j := \vec{\mathrm{S}}_j[0]$ was not closed, and so the oracle $r_j := \text{DLOG}(j)$ 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 two equations yields $s_i^* G = r_j G + \vec{\alpha}_j G + \vec{\beta}_j x G + \mathsf{H}(R_i^*, X, m_i^*)xG$, and thus $s_i^* \equiv_q r_j + \vec{\alpha}_j + \vec{\beta}_j x + \mathsf{H}(R_i^*, X, m_i^*)x$. From this, L computes and returns $r_j := \log_G(R_j)$ and thereby wins game wOMDL, since r_j is

the discrete logarithm of a challenge that was not solved by the oracle DLog, as required by the game.

Therefore we obtain:

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

Now let q be an upper-bound of successfully closed sessions via queries to the $\textsc{Sign}_2$ oracle made by A. Then L's number of queries to its DLog oracle is also bounded by q, and by applying Lemma 1 we obtain an adversary D playing in the game DL where

$$\Pr[\mathsf{G}_1^{\mathsf{A}}(\lambda) \wedge \mathsf{E}] \leq q \cdot \exp(1) \cdot \mathsf{Adv}^{\mathsf{DL}}_{\mathsf{GrGen},\mathsf{D}} \ . \tag{16}$$

$\underline{\mathsf{G}_3}$. 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 $sk := (q, \mathbb{G}, G, \mathsf{H}, x)$ 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 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 $(\bar{s} - \alpha) \bmod q$ to the list $\vec{\mathrm{D}}$. In $\textsc{Sign}_2$ instead of returning $s := (r + cx) \bmod q$ 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 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 $c \equiv_q \mathsf{H}(R + \alpha G + \beta X, X, m) + \beta$. We show that for any choice of α, β, message m and signing key sk, the user's view in G_3 is distributed equivalently to its view in G_2. The latter is

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

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

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

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

$$\Pr[\mathsf{G}_2^{\mathsf{A}}(\lambda)] = \Pr[\mathsf{G}_3^{\mathsf{A}}(\lambda)] \ . \tag{17}$$

$\underline{\text{Reduction of sEUF-CMA of Schnorr to } \mathsf{G}_3.}$ 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 $\Pr[\mathsf{G}_3^{\mathsf{A}}(\lambda)]$.

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 Sign. With the Schnorr parameters sp it completes 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 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 Sign. The simulation is perfect.

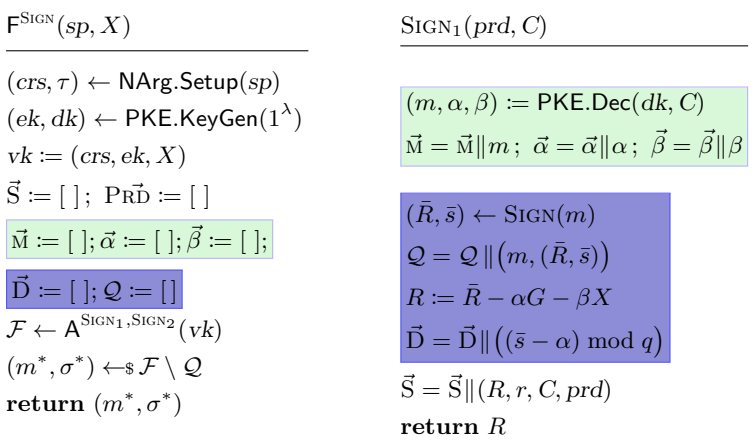

Fig. 8. F paying against sEUF-CMA security of $\mathsf{Sch}[\mathsf{GrGen}, \mathsf{HGen}]$. The oracle SIGN_2 is simulated to A is as defined in game G_3 in Figure 2.

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 $\mathcal{Q} = (m_j, \sigma_j := (\bar{R}_j, \bar{s}_j))_{j \in [|\vec{\mathrm{S}}|]}$ (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 $\mathcal{Q}_{\text{cls}} := (m_j, \sigma_j)_{j \in J}$.

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 an injective function $f\colon [n] \to J$ mapping elements of $\mathcal{F}$ to elements of $\mathcal{Q}_{\text{cls}}$, in particular, $m_i^* = m_{f(i)}$. 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 $1 = \vec{\mathrm{PRD}}_{f(i)}(m_{f(i)}) = \vec{\mathrm{PRD}}_{f(i)}(m_i^*)$ for all $i \in [n]$, which contradicts the winning condition of G_3, which requires that no such f 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 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:

$$\Pr[\mathsf{G}_3^{\mathsf{A}}(\lambda)] \leq \mathsf{Adv}^{\mathsf{sEUF\text{-}CMA}}_{\mathsf{Sch}[\mathsf{GrGen},\mathsf{HGen}],\mathsf{F}}(\lambda)\ . \tag{18}$$

Theorem 1 now follows from Equations (13) and (15)–(18). □

Remark 1. If we considered the weaker definition of unforgeability obtained by moving $\vec{\mathrm{PRD}} = \vec{\mathrm{PRD}} \| prd$ from SIGN_2 to SIGN_1, the predicates of all opened sessions are included in $\vec{\mathrm{PRD}}$.

The argument in the 3rd-to-last paragraph in the above proof would then directly yield that there exists an element $(m_{i^*}^*, \sigma_{i^*}^*) \in \mathcal{F}$ that is not in $\mathcal{Q}$ (i.e., all sessions and not only the closed ones in $\mathcal{Q}_{\text{cls}}$). Since the argument that $(m_{i^*}^*, \sigma_{i^*}^*) \notin \mathcal{Q} \setminus \mathcal{Q}_{\text{cls}}$ is therefore no longer

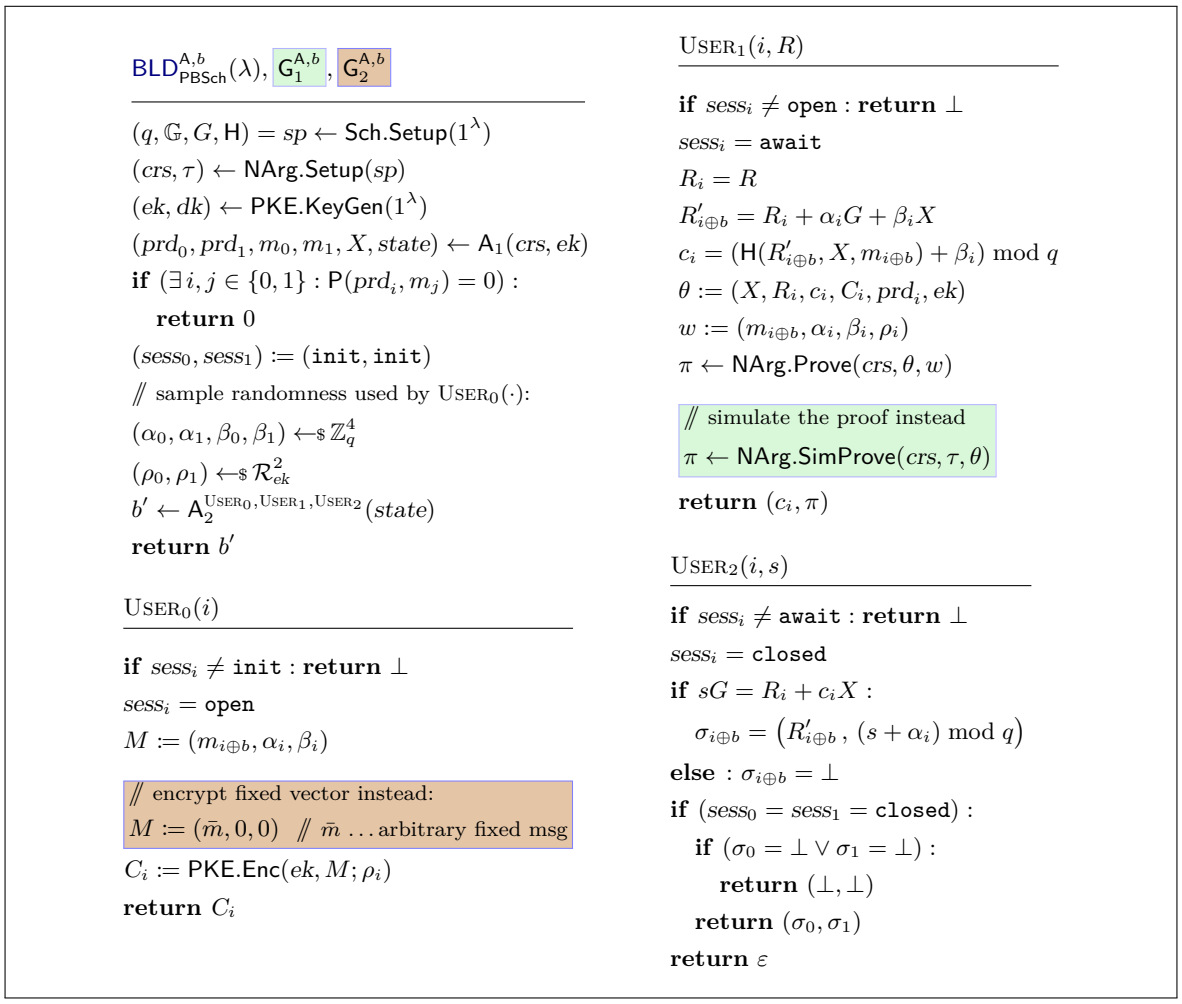

Fig. 9. The blindness game from Figure 3 for the scheme $\mathsf{PBSch}[\mathsf{P}, \mathsf{GrGen}, \mathsf{HGen}, \mathsf{PKE}, \mathsf{NArg}]$ from Figure 4 (ignoring all boxes) and hybrid games used in the proof of Theorem 2. G_1 includes the light green box and G_2 includes both boxes.

required, neither is the abort condition E and thus the game hop from G_1 to G_2. This means that the last term in the security bound of Theorem 1 vanishes.

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.

$\underline{\mathsf{G}_0}$. This is game BLD from Figure 3 with PBS instantiated with PBSch from Figure 4, that is, 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 NArg.Prove we use the simulator 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 $\mathsf{NArg}[\mathsf{R}_{\mathsf{Sch}}]$.

$\mathsf{Z}_b^{\textsc{Prove}}(crs)$

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

$\textsc{User}_1(i, R)$

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

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

According to the definition of game ZK, Z_b receives as input crs generated by NArg.Setup on input sp generated by NArg.Rel, which is defined as 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 $\Pr[\mathsf{ZK}^{\mathsf{Z}_b,0}_{\mathsf{NArg}[\mathsf{R}_{\mathsf{Sch}}]}] = \Pr[\mathsf{BLD}^{\mathsf{A},b}_{\mathsf{PBSch}}]$, and $\Pr[\mathsf{ZK}^{\mathsf{Z}_b,1}_{\mathsf{NArg}[\mathsf{R}_{\mathsf{Sch}}]}] = \Pr[\mathsf{G}_1^{\mathsf{A},b}]$ and therefore

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

Together with the triangular inequality this yields:

$$\begin{aligned}\mathsf{Adv}^{\mathsf{BLD}}_{\mathsf{PBSch},\mathsf{A}} &:= \left|\Pr[\mathsf{BLD}^{\mathsf{A},1}_{\mathsf{PBSch}}] - \Pr[\mathsf{BLD}^{\mathsf{A},0}_{\mathsf{PBSch}}]\right| \\ &= \left|\Pr[\mathsf{BLD}^{\mathsf{A},1}_{\mathsf{PBSch}}] - \Pr[\mathsf{G}_1^{\mathsf{A},1}] + \Pr[\mathsf{G}_1^{\mathsf{A},1}] - \Pr[\mathsf{BLD}^{\mathsf{A},0}_{\mathsf{PBSch}}] + \Pr[\mathsf{G}_1^{\mathsf{A},0}] - \Pr[\mathsf{G}_1^{\mathsf{A},0}]\right| \\ &\leq \mathsf{Adv}^{\mathsf{ZK}}_{\mathsf{NArg}[\mathsf{R}_{\mathsf{Sch}}],\mathsf{Z}_1} + \left|\Pr[\mathsf{G}_1^{\mathsf{A},1}] - \Pr[\mathsf{G}_1^{\mathsf{A},0}]\right| + \mathsf{Adv}^{\mathsf{ZK}}_{\mathsf{NArg}[\mathsf{R}_{\mathsf{Sch}}],\mathsf{Z}_0} . \quad (19)\end{aligned}$$

$\underline{\mathsf{G}_2}$. 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.

$\mathsf{C}_b^{\textsc{Enc}}(ek)$

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

$\textsc{User}_0(i)$

if $sess_i \neq \texttt{init}$: **return** $\perp$
$sess_i = \texttt{open}$

$M_0 := (\bar{m}, 0, 0)$
$M_1 := (m_{i \oplus b}, \alpha_i, \beta_i)$
$C_i \leftarrow \textsc{Enc}(M_0, M_1)$
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 $q, \mathbb{G}, G, \mathsf{H}$ 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:

$$\begin{aligned}\left|\Pr[\mathsf{G}_1^{\mathsf{A},1}] - \Pr[\mathsf{G}_1^{\mathsf{A},0}]\right| &= \left|\Pr[\mathsf{G}_1^{\mathsf{A},1}] - \Pr[\mathsf{G}_2^{\mathsf{A},1}] + \Pr[\mathsf{G}_2^{\mathsf{A},1}] - \Pr[\mathsf{G}_2^{\mathsf{A},0}] + \Pr[\mathsf{G}_2^{\mathsf{A},0}] - \Pr[\mathsf{G}_1^{\mathsf{A},0}]\right| \\ &\leq \mathsf{Adv}_{\mathsf{PKE},\mathsf{C}_1}^{\mathsf{CPA}} + \left|\Pr[\mathsf{G}_2^{\mathsf{A},1}] - \Pr[\mathsf{G}_2^{\mathsf{A},0}]\right| + \mathsf{Adv}_{\mathsf{PKE},\mathsf{C}_0}^{\mathsf{CPA}}\ . \quad (20)\end{aligned}$$

$\underline{\text{Reducing } \mathsf{G}_2 \text{ to perfect blindness of "plain" Blind Schnorr.}}$ 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 (19) and (20), concludes the proof. □

D On Alternative Constructions of PBS

While trusted parameters can be avoided in practice by assuming the random-oracle model (as we discuss in Section 5.1), one might wonder whether we can directly instantiate our blueprint using building blocks without parameters. (So blindness would automatically hold against signers that set up the system.)

Zaps. Without increasing the round-complexity of the signing protocol, we cannot use NIZK proofs [GO94], but could replace NArg by a *zap* [DN07], which is a witness-indistinguishable (WI) proof system without parameters. However, when relying on WI only, the first user message in the signing protocol (C in Figure 4) must perfectly hide its content. (Since proofs cannot be simulated, in the blindness game there must exist two witnesses that explain C as containing either m_0 or m_1.) This precludes the use of an encryption scheme.

Extractable commitments. In a parameter-free setting, we cannot use public-key encryption, but could use a commitment for the first user message (which would have to be perfectly hiding when using it with a zap). Since in the proof of unforgeability we need to extract the committed value, we would need a *knowledge commitment* [Gro10] (or combine a commitment with a (parameter-less) proof of knowledge).

In either case we would have to resort to (strong) extractability assumptions. That is, for any adversary B that returns a commitment C, there exists an extractor E, which on input B's internal randomness, returns a committed value and randomness that yields C. We moreover need to assume *auxiliary input* for B, which B can use in the computation of C, and which is also given to E.

For an adversary A in game UNF, we can define B_1, which on ("auxiliary") input the Schnorr parameters and key X simulates UNF for A and stops at A's first call to $\textsc{User}_1$ and returns A's value C. For B_1 there exists an extractor E_1, which the reduction R for unforgeability runs (on A's randomness and its own input) to obtain the committed value (m_1, α_1, β_1). Then R queries m_1 to its signing oracle and uses the reply $(\bar{R}_1, \bar{s}_1)$ to answer A's query.

Now to extract from A's second signing query, we would have to define B_2, which however needs to answer A's first query, for which it would have to run E_1 to obtain m_1 and needs $(\bar{R}_1, \bar{s}_1)$ as auxiliary input.

The two issues with this approach are:
1) Every adversary B_i needs to run the extractors $\mathsf{E}_1, \ldots, \mathsf{E}_{i-1}$ to extract the messages $m_1, \ldots, m_{i-1}$. Even if E_i ran in the same time as B_i, still B_i would run in time exponential in i, and thus R would not be efficient.[20]

[20] Let $t_{\mathsf{A},i}$ be A's running time until the i-th signing query. Let $t_{\mathsf{B},i}$ be B_i's running time, which is $t_{\mathsf{A},i}$ plus the running time of E_j for $j = 1 \ldots i-1$. Since we assumed E_j runs in time $t_{\mathsf{B},j}$, we have $t_{\mathsf{B},i} := t_{\mathsf{A},i} + \sum_{j=1}^{i-1} t_{\mathsf{B},j} = t_{\mathsf{A},i} + \sum_{j=1}^{i-1} 2^{i-j-1} t_{\mathsf{A},j}$.

2) The second issue is that the auxiliary input for B_i (signatures $(\bar{R}_j, \bar{s}_j)_{j\in[i-1]}$) depends on $(m_j)_{j\in[i-1]}$. We would thus have to assume extractability in the presence of auxiliary input whose distribution depends on the adversary's randomness, necessitating a stronger extractability definition.

E Further Discussion of Hardwiring

Minimal hardwiring refers to using the relation $\mathsf{R}_{\mathsf{Sch}}$ as defined in Section 4, that is, with $par_{\mathsf{R}} = (q, \mathbb{G}, G, \mathsf{H})$ and statements of the form $\theta = (X, R, c, C, prd, ek)$. The same CRS, and thus scheme parameters par_{R}, can thus be used by multiple signers since they are independent of their signature verification keys X.

If a non-transparent scheme, such as *Groth16* is used, par_{R} should be set up in a "ceremony" using multiparty computation [BGM17, KMSV21], since the signer's security relies on the secrecy of the simulation trapdoor. On the other hand, due to subversion zero knowledge of *Groth16*, the users need not trust the ceremony to obtain blindness if they perform a (potentially complex) CRS-consistency check [Fuc19]. But since the CRS can be used by many signers, users can expect that a malformed CRS would be recognized and reported quickly. They can therefore optimistically not check the CRS themselves.

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 $\mathsf{R}_{\mathsf{Sch}}{}'$ (cf. Eq. (7)) with modified syntax $par_{\mathsf{R}'} = (q, \mathbb{G}, G, \mathsf{H}, X, ek)$ and $\theta = (R, c, C, prd)$, which yields performance gains in terms of CRS size, as well as prover and verification time. 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 $par_{\mathsf{R}'}$ (cf. Eq. (7)) 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 $\mathsf{R}_{\mathsf{Sch}}{}'$ runs $(q, \mathbb{G}, G, \mathsf{H}) \leftarrow \mathsf{Sch.Setup}(1^\lambda)$ (as for $\mathsf{R}_{\mathsf{Sch}}$), and in addition $(ek, dk) \leftarrow \mathsf{PKE.KeyGen}(1^\lambda)$; $x \leftarrow\!\!\$\, \mathbb{Z}_q$ and $X := xG$. 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.

F Probabilistic Verification of a Groth16 CRS

Verifying the well-formedness of a CRS (proving key) pk in *Groth16* is done by evaluating (and comparing) pairings $e\colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ on components $pk_1, pk_2, \ldots, pk'$ from the groups $\mathbb{G}_1$ and $\mathbb{G}_2$ of the key [Fuc18]. Since for many sets of pairings, one of the arguments is the same, we can use probabilistic batch-verification techniques [FGHP09, BFI+10] to speed up computation considerably.

For example consider a set of equations $LHS_i := e(\sum_{j=1}^{d} a_{i,j} pk_j, pk') \stackrel{?}{=} RHS_i$ for $i \in [n]$ (where $a_{i,j}$ are from the relation description). Instead of checking each equation individually, we choose $r \leftarrow\!\!\$\, \mathbb{F}_p$ and check whether $\prod_i LHS_i^{r^i} = \prod_i RHS_i^{r^i}$. (By the Schwartz-Zippel lemma (a.k.a., the polynomial identity lemma), if this equation holds then with all but negligible probability over the choice of r, all individual equations hold.) By bilinearity of e, we have

$$\prod_i LHS_i^{r^i} = e\Big(\sum_{j=1}^{d} \big(\sum_{i=1}^{m} r^i a_{i,j} \big) pk_j, pk' \Big) ,$$

whose computation requires $m \cdot d$ multiplications (and m additions) in $\mathbb{F}_p$ and a multiscalar multiplication (MSM)[21] of size d in $\mathbb{G}_1$ as well as 1 pairing.

This reduces the computation in the consistency check in [Fuc18] from $(7d + 4m + 2)$ pairings and $m \cdot (2 \cdot \mathsf{MSM}_1^d + \mathsf{MSM}_2^d) + \mathsf{MSM}_1^d$ multiscalar multiplications (where d is the number of gates and m the number of wires of an R1CS instance) to 15 pairings and $3 \cdot \mathsf{MSM}_1^d + \mathsf{MSM}_2^d + 3 \cdot \mathsf{MSM}_1^m + \mathsf{MSM}_2^m$ MSMs, as well as $3 \cdot m \cdot d$ multiplications in $\mathbb{F}_p$.

The "Proving key verification" times we list in Table 1 are estimated based on benchmark results for BN254 (*go* implementation of ConsenSys) conducted by [Hou21]. We estimate the cost of one multiplication in the base field to be 1/10-th the cost of a group operation. For the MSM problem we take the runtime asymptotics of the algorithm from [BDLO12], which is a modification of Pippenger's multi-scalar-multiplication method [Pip76]. We assume that our number of constraints d is equal to m (which is approximately correct for most R1CS instances). We further assumed full parallelizability of our probabilistic proving key checking algorithm and hence bluntly divided our results by 12, which is the number of cores we used for our experiments.

G A NIZK with secp256k1 Support

Most NIZK systems have limited flexibility in terms of the elliptic curves for which they support efficient proving of the group operation. Moreover, there is no best practice on how to construct a NIZK for a particular curve. We therefore describe some (out of possibly many) ways of obtaining a NIZK with efficient support for the curve secp256k1. Due to the many uses of that curve, this is of great practical interest and relevance. The curve secp256k1 is defined by the equation $\mathcal{E}\colon y^2 = x^3 + 7$ 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.

Pairing-based NIZK schemes (such as [Gro16, GWC19, SL20]) are not ideal candidates, since there is no known pairing-friendly curve with a subgroup of size p. One therefore has to pay a factor of up to 1000 overhead in order to simulate operations in $\mathbb{F}_p$ in a different field [SSS+22]. This overhead can be seen in our benchmarks where we emulate $\mathbb{F}_p$ arithmetic in the scalar field of the BN254 curve (scenarios (B1) and (B2) in Table 1).

On the other hand, curves with a subgroup of size p can be created relatively easily using the Cocks-Pinch method.[22]. The downside is that the base field over which the curve is defined has around twice the bit-length of the subgroup order [FST10].

[21] MSM denotes the problem of computing $S = \sum_i a_i G_i$ for coefficients (a_i) and group elements (G_i). For k elements from $\mathbb{G}_i$ we denote this MSM_i^k.

[22] This method by Clifford Christopher Cocks and Richard G.E. Pinch was never published, however it is frequently miss-cited as having appeared in [BF01].

An alternative to pairing-based schemes starts with Silverman and Stange's observation [SS11] that the secp256k1 curve has a "twin" defined over the same equation, which is called secq256k1 and whose order is p. While this is not a pairing-friendly curve, this "cycle of curves" [BCTV14] suggests the use of a NIZK that does not require pairings and can be instantiated over the secq256k1 curve, such as *Bulletproofs* [BBB+18] or *Halo* [BGH19]. This way, one avoids the large overhead of simulating mod-p arithmetic and hence group operations of the curve secp256k1 can be arithmetized very efficiently.

Another challenge comes from the prime factors of the group order

$$|\mathbb{F}_p^*| = p - 1 = 2 \cdot 3 \cdot 7 \cdot 13441 \cdot \\ 0\text{x}1\text{db}8260\text{e}5\text{e}3\text{b}460\text{a}46\text{a}0088\text{fccf}6\text{a}3\text{a}5936\text{d}75\text{d}89\text{a}776\text{d}4\text{c}0\text{da}4\text{f}338\text{aafb} \; . \quad (21)$$

They imply that there is no support for standard run-time-optimal FFT techniques (which are needed by the vast majority of NIZKs) but requires 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.[23] On the other hand, $13441 - 1 = 2^7 \cdot 3 \cdot 5 \cdot 7$ has small factors. This enables the use of an FFT technique by Rader [Rad68] and hence for at least this particular subgroup, which is still large enough to arithmetize the relation for a decent size of message and predicate complexity (as in scenario (A) in Table 1), we have support for almost run-time-optimal FFT techniques again.

Transparent-setup zkSNARKs such as *Redshift* [KPV19], *STARKs* [BBHR19], *Aurora* [BCR+19], or *Plonky2* [Zer22], which forgo the use of elliptic curves, face a similar problem. For committing to the circuit assignment, they all use the IOPP protocol FRI [BBHR18], which has to be instantiated over $\mathbb{F}_p$ in order to avoid expressing the modulo-p reduction explicitly. 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. (21). To the best of our knowledge, it remains open whether FRI (which shares many features of the FFT algorithm) can be efficiently executed over non-smooth domains.

As a candidate NIZK to implement our predicate blind Schnorr signature scheme with efficient secp256k1 support we suggest *Halo 2* [Com21], an extension of *Halo*. *Halo 2* also has a transparent setup and can be instantiated over the curve cycle secp256k1–secq256k1. It supports batch verification, recursive proof composition via proof accumulation and, due to its "*Plonk*-ish arithmetization" [GWC19], it supports lookups. These become particularly useful when multiple invocations of binary operations (as is the case for SHA-256) are required. Moreover, proof accumulation 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 secp256k1–secq256k1.

[23] 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 impractical.