Concurrently Secure Blind Schnorr Signatures

Georg Fuchsbauer and Mathias Wolf

TU Wien, Austria
first.last@tuwien.ac.at, m.last@mailfence.com

24 May 2024

Abstract. Many applications of blind signatures, e.g. in blockchains, require compatibility of the resulting signatures with the existing system. This makes blind issuing of Schnorr signatures (now being standardized and supported by major cryptocurrencies) desirable. Concurrent security of the signing protocol is required to thwart denial-of-service attacks.

We present a concurrently secure blind-signing protocol for Schnorr signatures, using the standard primitives NIZK and PKE and assuming that Schnorr signatures themselves are unforgeable. Our protocol is the first to be compatible with standard Schnorr implementations over 256-bit elliptic curves. 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 implementations and benchmarks for various choices of primitives and scenarios, such as blindly signing Bitcoin transactions only when they meet certain conditions specified by the signer.

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], *private access tokens* [HIP+] and *Privacy Pass* [DGS+18]. 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 be a stringent requirement: changing the supported signature schemes for blockchain systems requires consensus, which is a lengthy process; moreover, as participants update their client software asynchronously, updates must be backwards compatible (i.e., soft forks) to avoid a segregation of the network.

Schnorr signatures. One of the most important signature schemes today are Schnorr signatures [Sch90]. Since their patent expired in 2008, they have been outpacing RSA signatures in application counts. Schnorr signatures are much smaller and more efficiently verifiable for a comparable security level. (EC)DSA, a NIST-standardized signature scheme, has efficiency comparable to Schnorr, but it requires unrealistic idealizations to be proved secure [FKP16, FKP17, HK23]. 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 that treats cryptographic hash functions 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 *Mimblewimble* [Poe16, FOS19, FO22] crucially relies on. Standardization and wide-spread usage makes blind signature schemes that produce Schnorr signatures desirable, but using Schnorr can also be a necessity: for example, blind coin swaps using *scriptless scripts* [Nic19] in Bitcoin and potential applications we discuss below all require blind issuing of Schnorr signatures.

Blind Schnorr signatures. Schnorr signatures admit an 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 this case, the signer can only engage in a signing session once the previous session has been finished or canceled, which opens the door to denial-of-service (DoS) attacks. This motivated the development of concurrently secure blind signature schemes, where the adversary is allowed to interweave several signing sessions [Bol03, BNPS03, Oka06, KZ06, HKKL07, HKLN20, KLR21, KLX22, CAHL$^+$22, TZ22, HLW23], or even *round-optimal* schemes [Fis06, AFG$^+$10, FV10, GRS$^+$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. He also showed that solving ROS 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, 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.

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

[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 $X_1, \dots, X_n$, the adversary returns a group element Z, one can extract a *representation* $(\zeta_1, \dots, \zeta_n)$ so that $Z = \sum \zeta_i X_i$.

signer and the user run two parallel signing sessions of which the signer finishes only one picked at random. They prove unforgeability in the algebraic group model and the ROM from the *one-more discrete logarithm* (OMDL) assumption (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 reduces the security of their scheme to 70-bit for standard instantiations of Schnorr.[3]

Security of the original blind Schnorr signature scheme [CP93] when signing sessions are only performed sequentially was shown 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.[4]

Garg et al. [GRS$^+$11] construct a round-optimal blind-signing protocol for any signature scheme. They "stress that [their] result is only a feasibility result", as it makes use of complexity leveraging, leading to a "signature size of hundreds of kB" [HK16]. Moreover, the signing protocol uses Yao's garbled circuits [Yao82], which are another source of inefficiency.

The lack of concurrently secure blind signing protocols for Schnorr signatures with standard parameters 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, 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 can be too strong and the signer might want to control parts of the message. This is what "partially" blind signatures [AF96, AO00] provide. In this 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 on the one hand "Schnorr-like" schemes (which do not require pairings): Tessaro and Zhu [TZ22] build on blind Schnorr [FPS20] and obtain a scheme with signatures in $\mathbb{G} \times \mathbb{Z}_p^3$ and a proof from DL in the AGM+ROM (where p is the order of the underlying group $\mathbb{G}$), which was recently improved to $\mathbb{G} \times \mathbb{Z}_p^2$ [CKM$^+$23]; 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 [HLW23] improve on recent techniques [KLR21, CAHL$^+$22] for a pairing-based scheme [Bol03] proven from (co-)CDH in the ROM, with round-optimal issuing [Fis06] but relatively large signatures.[5]

[3] The authors propose to generalize their scheme to $t > 2$ parallel runs (of which the signer finishes one). But even assuming the best attack on mROS is guessing which sessions will be finished, for 128-bit security we would require $t > 2^{14}$ parallel sessions, resulting in huge communication complexity.

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

[5] A signature in [HLW23] consists of a BLS signature [BLS01], and $K - 1$ keys and commitment openings. For 128-bit security the authors suggest $K = 33$, yielding 5.71 kB per signature, compared to 128 bytes for [TZ22].

Our contributions

We present the first practical concurrently secure blind and partially blind signing protocol for issuing standard Schnorr signatures with rigorous security guarantees. Our blind-signing protocol 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.[6]

<u>Overview of our scheme.</u> Our starting point is the original 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 samples two possible sets of "blinding values" (which represent the user's randomness during a session). The signer's first protocol message, a group element R, will then determine which set of blinding values the attacker will use in every session to compute the forgeries. The crucial observation is that before receiving R, the attacker does not know which blinding values it will use.

Attempting to prevent this specific attack, we could 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 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.[7] It turns out that 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 in Theorem 1.

Observe that when the user sends a proof that her protocol message is consistent with the message to be signed, she might additionally 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). Moreover, as shown by our benchmarks, this can come at very little computational cost.

Concretely, our construction uses a public-key encryption scheme PKE for the "commitment" in the first round[8] 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 standard chosen-plaintext security.

[6] 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 $\log q \leq 126 - n$, 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]).

[7] 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, CAHL+22, HLW23]. 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).

[8] This enables "straight-line" extraction of the committed values during the signing queries in our proof of unforgeability. We cannot use commitments that assume non-blackbox extraction: the reduction would have to run extractors that run other extractors, which would lead to an exponential blow-up of its running time. Moreover, the efficiency gains in our implementation would be small. (See Appendix D for a detailed discussion.) Note that replacing the NIZK by a proof of knowledge would not help since we need to extract before the proven statement is known.

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 this is a non-standard assumption, it is a minimal assumption in any scenario that uses Schnorr signatures (and for Schnorr instantiations such as using curve secp256k1 and SHA-256 it is arguably uncontroversial). We make this assumption because the statement proved by the NIZK scheme involves the hash function used by the signature scheme, so we cannot rely on the security of Schnorr signatures in the random oracle model.[9]

The security of our scheme thus relies solely on the security of its building blocks and we do not make any additional assumptions, such as OMDL or (variants of) ROS, nor work in idealized models. Viewed differently, adding our blind-signing protocol to an application already using Schnorr signatures only requires additionally assuming standard security of PKE and NArg.

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, by working in 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).

As proof system, 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) [BBHR18b, BBB+18, BCR+19, BFS20, BGH19, COS20, Set20, SL20, Com21, Zer, KPV22, BC23, CBBZ23]. As the proof system NArg is the only computationally complex part of our construction, we give a prototype implementation using the transparent-setup NIZK *Spartan* [Set20].

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, then still holds. *Groth16* [Gro16], the zk-SNARK with the shortest proofs, has been shown to satisfy this notion [Fuc18] if the prover first performs a consistency check on the NIZK parameters. The users would have to perform this check once.[10] In practice, users could also optimistically trust the signer, since the discovery of the inconsistency of her parameters would harm her reputation. We implemented and benchmarked NArg using *Groth16*.

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). We implemented and benchmarked NArg using *Plonk*.

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]

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

[10] We analyze the computational complexity of this check in Appendix F.

derive the randomness $r = \log R$ used during signing deterministically, by hashing the message and the signer's secret key.[11] 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 [AF96] and improves on the privacy guarantees. While in partially blind signatures, the signer agrees with the user on the public part of the messages before blindly signing it, 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 generalizes to predicates that take a witness as additional input, so that NP-statements can be enforced 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 want to protect customer privacy, while only authorizing transactions compliant with the law or internal rules, like limiting the amount of a transaction to certain countries or individuals. Using PBS, the provider learns neither amount nor destination, only that the criteria are met.

Realizing this with partially blind signatures would require stating the conditions explicitly in the public message part. Signature verification would be more cumbersome, since it is left to the verifier to check whether the message conforms to its public part. Worse, the signature would reveal the conditions, which remain hidden when using PBS.

We give a (concurrently secure) instantiation of PBS whose signatures are standard Schnorr signatures. As these are supported by Bitcoin [WNR20], this enables concurrently secure blind coin swaps [Nic19]. But using the "predicate" functionality also opens up new applications, like adding anonymity to payments for users that entrust their coins to a cryptocurrency exchange. In this scenario, the user can construct a payment from the exchange's address and has it blindly signed under a predicate that enforces (an upper-bound on) the paid amount; the exchange then debits the user's account by the amount stated in the predicate. When the user posts the transaction on the blockchain, the exchange cannot link it to the user (it only knows it cannot be one transferring more than the agreed amount). This is one of the scenarios considered in our implementations.

But the (NP-)predicate can also encode further restrictions, like paying limits depending on the user's credentials (while preserving anonymity), or compliance with the law.[12] We view PBS as a means to reconcile privacy and compliance and our construction is compatible with a number of existing systems.

[11] 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].

[12] Another potential application (not supported by partially blind signatures) is to rate-limiting in Privacy Pass [DGS+18]: when obtaining the signed tokens, PBS could enforce that they are "linked" among them (but unlinkable to the signing session), so that applications can enforce rate limits on linked tokens. (E.g., a (long-enough) prefix of the signed string must be the preimage of a one-way-function evaluation that specifies the predicate used during blind signing.)

Implementations

To give estimates of the efficiency of our construction, we implemented the 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 three choices for NArg: (G) *Groth16* [Gro16] (*Iden3* implementation [ide]), a subversion-zero-knowledge SNARK requiring trusted parameters for soundness; (P) the "universal" zk-SNARK *PlonK* [GWC19] (*Fluidex* implementation [Plo]); and (S) a prototype by Tehrani and Sankar [TS] of the NIZK from the *Spartan* family [Set20], which does not require a trusted setup. We wrote the circuits for various scenarios 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 and the one-time pad in the prime field of the NIZK as DEM. As the underlying group, we use the *Baby JubJub* (BJB) curve group [BJB20] for (G) and (P) and secp256k1 for (S), and a sponge hash from the Poseidon family [GKR+21]. (These choices are motived by their concise circuit representations.)

We first consider scenarios in which the underlying Schnorr instantiation also uses BJB and Poseidon. For both proof systems (G) and (P), the *proving key*, i.e., the (larger) 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 columns (A1)–(A3) in Table 1, page 25).

We then test our scheme for Bitcoin, that is, we use as Schnorr parameters the curve secp256k1 and SHA-256 [WNR20]. These are not efficiently "arithmetizable" in the used configurations of (G) and (P)[13], therefore performance is worse, but still practical. The CRS is now up to 550 MB and proofs take around 1 (G) or 3 minutes (P) to generate, while their size does not increase; verification time also remains unchanged compared to the optimized scenario. The computational burden on the signer's side, like the cryptocurrency exchange in the above example is thus small.[14] As a scenario for "predicate-blind" signing we consider signing a Bitcoin transaction when certain parts of the transaction (such as an upper bound on the amount) are fixed by the signer (row (B2) in Table 1). Compared to "fully-blind" signing, the CRS size and running times hardly change.

Finally, we give an outlook on how a proving system like (S), which interoperates well with the secp256k1 curve compares to (G) and (P). Another advantage of (S) is that it does not require a trusted setup. While the CRS size reduces to merely 36 kB (cf. Table 2), surprisingly, with 2.5 minutes, the proving time is still comparable to (P). We conjecture that this is largely due to the fact that the used prototype implementation [TS] does not, as far as we could tell, leverage the huge potential for parallelization, nor does it optimize arithmetic in the "outside" curve secq256k1 or the "inside" curve secp256k1.

Despite a lot of potential for improving efficiency, we stress that privacy-preserving applications that require minutes of computation are not uncommon in the blockchain space. For example, computing a "private" transaction in the first generation of *Zcash* took around two

[13] We implement both (G) and (P) over the BN254 curve [BN06], whose order is incompatible with the base field of secp256k1.

[14] 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 ed25519 is also incompatible with the BN254 scalar field.

minutes[15] and their *proving key* was 868 MB.[16] The numbers we report should be viewed as very rough upper bounds, as the implementations are far from optimized, the selected NIZKs are not leading in terms of performance and results depend on machine specifics (e.g., proof verification for (P) was reported [Bot] as 100 times faster than measured by us).

In Table 1 we also 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).

Efficiency of Generic Schemes. Blind signing for any scheme can be implemented using generic two-party computation between signer and user [JLO97] (which would not be concurrently secure [HKKL07]). A generic technique are garbled circuits (GC) [Yao82], which are also used in the round-optimal construction by Garg et al. [GRS+11]. As a rough estimate of the efficiency of Schnorr blind signing using GC, we consider Jayaraman, Li and Evans' [JLE17] work, who use garbled circuits 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. Although GC have been shown to outperform custom protocols in other contexts [HEK12], for blind Schnorr signing their efficiency appears to be several magnitudes worse than our approach.

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.

We denote sets by calligraphic capital letters, e.g. $\mathcal{A}, \mathcal{B}, \mathcal{C}$, and algorithms by $\mathsf{Sans\ Serif}$ 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 $a \equiv_q b$.

[15] https://electriccoin.co/blog/software-usability-and-hardware-requirements/

[16] https://download.z.cash/zcashfinalmpc/sprout-proving.key

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*

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

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

$\underline{\mathsf{DL}^{\mathsf{A}}_{\mathsf{GrGen}}(\lambda)}$
$(q, \mathbb{G}, G) \leftarrow \mathsf{GrGen}(1^\lambda)$
$x \leftarrow\!\!\$\ \mathbb{Z}_q\,;\ \ X \coloneqq xG$
$y \leftarrow \mathsf{A}(q, \mathbb{G}, G, X)$
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* $\mathsf{NArg[R]}$ *is* ***(adaptively) computationally sound*** *if for every adversary* A

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

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

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

$par_{\mathsf{R}} \leftarrow \mathsf{NArg.Rel}(1^{\lambda})\,;\ (crs,\tau) \leftarrow \mathsf{NArg.Setup}(par_{\mathsf{R}})$

$(\theta,\pi) \leftarrow \mathsf{A}(crs)$

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

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

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

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

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

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

$(crs,\tau) \leftarrow \mathsf{NArg.Setup}(par_{\mathsf{R}})$

$b' \leftarrow \mathsf{A}^{\textsc{Prove}}(crs)$

return b'

$$\underline{\textsc{Prove}(\theta, w)}$$

if $\mathsf{R}(par_{\mathsf{R}},\theta,w) = 0$: **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***[17] *if for all* $\lambda \in \mathbb{N}$*:*

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

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

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:*

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

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

$\underline{\textsc{Sign}(m)}$

$\sigma \leftarrow \mathsf{Sig.Sign}(sk, m)$
$\mathcal{Q} = \mathcal{Q} \cup \{(m, \sigma)\}$
return σ

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

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

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

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

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

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

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

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

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

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 (this protects against certain *related-key attacks* [MSM+16]). 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 [AF96, AO00]. PBS define an interactive protocol that enables 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, the *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. 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., $vk = (par, key)$ and $par :\subseteq vk$.
- $\langle \mathsf{Sign}(sk, prd), \mathsf{User}(vk, prd, m)\rangle \to (b, \sigma)$: an interactive protocol with shared input par (implicit in sk and vk) and a predicate prd is run between the signer and user. The signer takes a 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:

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

  We write $(b, \sigma) \leftarrow \langle \mathsf{Sign}(sk, prd), \mathsf{User}(vk, prd, m)\rangle$ as 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* $\lambda \in \mathbb{N}$*:*

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

(Strong) unforgeability. For blind signatures this notion states that after the completion of n signing sessions, the user cannot compute $n+1$ distinct valid message/signature pairs. 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 $n+1$ pairs with 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

$\underline{\mathsf{UNF}^{\mathsf{A}}_{\mathsf{PBS[P]}}(\lambda)}$

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

$\underline{\mathrm{SIGN}_1(prd, msg)}$

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

$\underline{\mathrm{SIGN}_2(j, msg)}$

if $\vec{\mathrm{S}}_j = \varepsilon$ **then** // j-th session not open
$\quad$ **return** $\perp$
$(st, prd) \coloneqq \vec{\mathrm{S}}_j$
$(msg', b) \leftarrow \mathsf{PBS.Sign}_2(st, msg)$
if $b = 1$ **then**
$\quad \vec{\mathrm{S}}_j \coloneqq \varepsilon$ // close session j
$\quad \vec{\mathrm{PRD}} = \vec{\mathrm{PRD}} \| prd$ // store predicate prd
return msg'

Fig. 2. The strong **unforgeability game** for a predicate blind signature scheme $\mathsf{PBS[P]}$ with a 2-round signing protocol. $\mathsf{InjF}(\mathcal{A}, \mathcal{B})$ denotes the set of 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 $\forall\, i \neq j \in [n] : m_i^* \neq m_j^*$.

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, let ℓ be the number of closed signing sessions and prd_j the predicate for the j-th closed session. Then there exists an injective mapping $f \colon [n] \to [\ell]$ so that $\mathsf{P}(prd_{f(i)}, m_i^*) = 1$ for all $i \in [n]$.

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 when checking whether an attack was trivial; that is, opening and not finishing a session never prevents the adversary from winning.

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

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

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

In game UNF the adversary A gets a verification key vk as input and has access to two oracles SIGN_1 and SIGN_2. The oracles represent an honest signer and correspond to the two phases of the interactive protocol. 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)

$\mathsf{BLD}^{\mathsf{A},b}_{\mathsf{PBS[P]}}(\lambda)$

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

$\textsc{User}_0(i)$

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

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

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

$\textsc{User}_2(i, msg)$

if $sess_i \neq \mathtt{await}$ **then return** $\bot$
$sess_i = \mathtt{closed}$
$\sigma_{i\oplus b} \leftarrow \mathsf{PBS.User}_2(st_i, msg)$
if $(sess_0 = sess_1 = \mathtt{closed})$:
  if $(\sigma_0 = \bot \vee \sigma_1 = \bot)$:
    return $(\bot, \bot)$
  return (σ_0, σ_1)
// in case other session is still open:
return ε

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 $b = 1$.

used in successfully closed signing sessions (stored in the list $\vec{\textsc{Prd}}$), so that every message is mapped to a predicate it satisfies.[18]

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.

As in the *malicious-signer model* [Fis06], the adversary can choose its own verification key key (which together with the parameters constitutes vk). It also chooses two messages m_0 and m_1 as well as two predicates prd_0 and prd_1, which must both be satisfied by m_0 and m_1. The challenger chooses a bit b and runs the protocol as the user with the adversary, asking for a signature on message m_b for predicate prd_0 and then for m_{1-b} for predicate prd_1. Being given the resulting signatures on m_0 and m_1, the adversary must determine the bit b.

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

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

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

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

The adversary consists of two parts A_1 and A_2 of which A_1 outputs messages and predicates, a verification key part *key* and a state *st*. The experiment only continues if both messages satisfy both predicates. It runs the signing protocol with A_2 and acts as the user (modeled via three oracles $\textsc{User}_0, \textsc{User}_1, \textsc{User}_2$) 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 (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.

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 used predicate (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 $\big(info, (m_i^*, \sigma_i^*)_{i\in[n]}\big)$, 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 $\big(info, (m_i', \sigma_i)_{i\in[n]}\big)$, B returns $\big(m_i^* := (info, m_i'), \sigma_i\big)_{i\in[n]}$.

If A wins then all (m'_i, σ_i) are distinct and valid w.r.t. *info*; therefore all $((\mathit{info}, m'_i), \sigma_i)$ 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 $\vec{\mathrm{PRD}}$ (see Figure 2). For all $j \notin I$, we have $\mathsf{P}(\vec{\mathrm{PRD}}_j, m^*_i) = 0$ (cf. (1), since $\vec{\mathrm{PRD}}_j \neq \mathit{info}$ and $m^*_i = (\mathit{info}, m'_i)$). Since $|I| \leq n-1$, there is no injective function f with $\mathsf{P}(\vec{\mathrm{PRD}}_{f(i)}, m^*_i) = 1$ for all $i \in [n]$. Together, this means B has won UNF.

The definition of blindness by AO is similar but in the *honest-signer model* [JLO97], that is, their challenger samples the key pair $(\mathit{vk}, \mathit{sk})$ 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 $\mathit{info}_0 = \mathit{info}_1$; our adversary must output two messages (info_0, m'_0) and (info_1, m'_1) and two predicates satisfied by both messages, which implies $\mathit{info}_0 = \mathit{info}_1$. The reduction generates the key pair for the AO adversary and then simply relays the oracle calls.

4 Predicate Blind Schnorr Signatures

4.1 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 $r \leftarrow\!\!\$\ \mathbb{Z}_q$ and computes $R \coloneqq rG$, which it sends to the user. The user samples two *blinding values* $(\alpha, \beta) \leftarrow\!\!\$\ \mathbb{Z}_q^2$ and computes $R' \coloneqq R + \alpha G + \beta X$, which will be the first component of the blind signature. The user then computes the corresponding value $c' \coloneqq \mathsf{H}(R', X, m)$, blinds it as $c \coloneqq (c' + \beta) \bmod q$ and sends c to the signer. The signer replies with $s \coloneqq (r + cx) \bmod q$, which the user transforms to $s' \coloneqq (s + \alpha) \bmod q$ 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 concurrently secure, we require the user to first send an encryption C of m and the values α, β 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 the values m, α and β, and the signer will only send the final value s if this proof verifies. To obtain 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 $\mathsf{R}_{\mathsf{Sch}}$:

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

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

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.,

$$\begin{array}{l}\underline{\mathsf{NArg.Rel}(1^\lambda)} \\ (q, \mathbb{G}, G) \leftarrow \mathsf{GrGen}(1^\lambda) \\ \mathsf{H} \leftarrow \mathsf{HGen}(q) \\ \textbf{return } sp := (q, \mathbb{G}, G, \mathsf{H})\end{array}$$

Let GrGen be a group generation algorithm and HGen be a hash function generator (which together define $\mathsf{Sch.Setup}$; cf. 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 $par := (crs, ek)$.

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. 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 arguably uncontroversial in practice, given the wide-spread use of Schnorr signatures; and it is a *sine qua non* in any application involving Schnorr signatures anyway.

Theorem 1. *Let* P *be a predicate compiler and* GrGen *and* HGen *be a group and a hash generation algorithm; let* PKE *be a perfectly correct public-key encryption scheme; let* $\mathsf{Sch}[\mathsf{GrGen}, \mathsf{HGen}]$ *be the Schnorr signature scheme of Figure 1 instantiated with* GrGen *and* HGen*; and let* $\mathsf{NArg}[\mathsf{R}_{\mathsf{Sch}}]$ *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* $\mathsf{sEUF\text{-}CMA}$ *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*,*

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

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

$\underline{\mathsf{PBSch.KeyGen}(par)}$

$(q, \mathbb{G}, G) :\subseteq par$
$x \leftarrow\!\!\$\ \mathbb{Z}_q\,;\ X := xG$
$sk := (par, x)\,;\ vk := (par, X)$
return (sk, vk)

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

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

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

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

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 adversary 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 $\mathsf{R}_{\mathsf{Sch}}$ in (3), implies that $c \equiv_q \mathsf{H}(\bar{R}, X, m) + \beta$.

By the definition of Schnorr signing (Figure 1), letting $x \coloneqq \log X$ 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 $s \coloneqq (\bar{s} - \alpha) \bmod q$.

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 whether 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 a guessing argument following Coron [Cor00]). 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.

<u>FULL TIGHTNESS UNDER A WEAKER UNFORGEABILITY NOTION.</u> 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 $\vec{\text{PRD}} = \vec{\text{PRD}} \| prd$ 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 it 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* $\mathsf{R}_{\mathsf{Sch}}$ *from* (3)*. 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 zero knowledge of* $\mathsf{NArg}[\mathsf{R}_{\mathsf{Sch}}]$*, and*
- C_0 *and* C_1*, playing in game* CPA *against indistinguishability of* 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 $\mathsf{BLD}^{\mathsf{A},b}_{\mathsf{PBS}[\mathsf{P}]}$ for an arbitrarily fixed b, we first replace the user's proofs

π (in both signing sessions) by simulated proofs. We next replace the user's ciphertexts C by encryptions of a fixed message. These hops are indistinguishable by zero-knowledge of $\mathsf{NArg}[\mathsf{R_{Sch}}]$ 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.

<u>Alternative constructions.</u> In Appendix D we discuss the necessity of straight-line extraction (as provided by the use of a PKE), which precludes the use of non-blackbox extractable commitments. We also argue why we cannot replace the proofs π by *zaps* [DN07] (or *zaks* [FO18]), that is, witness-indistinguishable proofs (of knowledge) 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.

<u>Model.</u> The adaptations in the syntax and security definitions of $\mathsf{PBS}[\mathsf{P}]$ 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 $\mathsf{P}(\vec{\textsc{Prd}}_{f(i)}, m_i^*) = 1$ is replaced by

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

As for soundness of proof systems, 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).

<u>Construction.</u> The only change in the construction is that the prd-witness for m is now included in the witness for $\mathsf{R_{Sch}}$ 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 security proof would thus incur a security loss linear in the number of calls to $\textsc{Sign}_2$.

5 Design Choices, Implementation Details and Benchmarks

5.1 Avoiding a Trusted Setup

When defining security for predicate blind signatures (Definition 12 and 13), the parameters par are assumed to be generated in a trusted way, which in practice has to be dealt with. In our scheme PBSch (Figure 4), for a 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 signature issuing, 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, neither the signer nor the user should 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 user run $\mathsf{PKE.KeyGen}$ and the signer run $\mathsf{NArg.Setup}$. But the former is not practical, since typical applications would have a single signer and multiple users;[19] and the latter is potentially insecure (see below).

<u>PKE SETUP.</u> An alternative to all users creating their own encryption key is to generate a single *ek transparently*, that is, in a way so the corresponding secret key is not 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 by "hashing into the group" [BF01, BCI+10, WB19]: a fixed public string is hashed to obtain the public key. Public keys for *ElGamal* encryption [ElG85] or DHIES [ABR98], which we use to instantiate PKE in all our implementations, are group elements.

<u>NIZK SETUP.</u> As with PKE, we can also instantiate NArg with a scheme that has a transparent setup, of which there now exists many (see the citations on p. 5). A SNARK scheme with particularly short proofs 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 on soundness. On the other hand, for blindness the user relies on NArg being zero-knowledge, a property that also assumes a CRS that was set up in a trusted way.

This can be reconciled by using *subversion zero-knowledge* proof systems [BFS16]. These guarantee 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 a way to check if a CRS is well-formed. He shows that, under a "knowledge-type" assumption, if provers only accept a well-formed CRS, then subversion zero knowledge holds.[20] Once a CRS is checked (which only needs to be done once), the scheme can be used as before.

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 by variables, in both R1CS [BCR+19] 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* considered in Eq. (3), i.e.,

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

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

[20] There are attacks against *Groth16* in which proofs constructed under a malformed CRS leak information on the witness [CGGN17, Fuc19].

into *maximal hardwiring*:

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

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:

(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 argument 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 Bitcoin.

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) concerns blockchain protocols that add (predicate-)blind signing on top of existing Schnorr parameters. Standard hash functions like SHA-256 and elliptic curves like secp256k1 are typically not *arithmetic-circuit-friendly*, which increases the CRS size and proving time.[21]

5.4 Implementation

To assess the efficiency of our predicate blind signature construction PBSch from Section 4, we benchmark the NArg component, which is the computationally heavy component. For several variants of both scenarios (A) and (B) from Section 5.3 we consider three proof systems: The *Iden3* implementation [ide] of the trusted-setup zk-SNARK *Groth16* [Gro16]; the *Fluidex*[22] implementation [Plo] of the universal-setup zk-SNARK *PlonK* [GWC19]; and a prototype[23] [TS] of the transparent-setup NIZK *Spartan* [Set20]. We wrote the circuits in the domain-specific language of Circom 2.0 [ide] and made them available [mot].

[21] One invocation of the SHA-256 compression function requires around 26 000 R1CS (see Footnote 24) constraints [CGGN17, KPS18, ide].

[22] *Fluidex* builds on the [ide] code base and does not consider *PlonK*-specific optimization techniques such as [PFM+22].

[23] [TS] builds on the reference implementation of *Spartan* [Set] where the curve has been replaced by secq256k1 and combines this with a modified version of [Bha] which compiles Circom projects into the format required by *Spartan*.

In Tables 1 and 2 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.[24] For each proof system we report the time it takes to compute the proof (which includes computing the witness). As for *Groth16* and *PlonK* 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. All experiments were run on an Intel® Core™ i7-10850H CPU @ 2.70GHz × 12 with 31 GB of RAM.

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 Table 1. We propose and use a probabilistic verification of the "proving key" using batching techniques, which we discuss in Appendix F.

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). The field $\mathbb{F}_p$ is the base field of the curve *Baby JubJub* (BJB) [BJB20]. 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 BJB elements from the group $\mathbb{G}$ used by Schnorr, *we represent them in* roman *font*, e.g., the generator is G.

We instantiate *Spartan* over the curve secq256k1. This curve is sometimes referred to as "twin" of secp256k1, since they share the same curve equations and the order of one is the base field size of the other [SS11]. As a consequence, the arithmetization is over a 256-bit field and messages can be slightly larger.

To instantiate the encryption scheme PKE (used to encrypt α, β and m in Figure 4) we use the DHIES [ABR98] key encapsulation mechanism of ElGamal [ElG85] over the BJB 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 $\rho\mathrm{K}$. The ciphertext consists of the message blinded by this hash together with the element $\rho\mathrm{G}$.

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):[25]

[24] 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 "gate".

[25] Note that the first line in the return statement of (8) checks a congruence modulo q, whereas the third line requires modulo p. We stress the importance of type checks when inputs are not $\mathbb{F}_p$ elements; for elements of the statement θ these can be directly performed when verifying a NIZK proof, which saves on CRS size and prover time.

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 $\mathsf{R}_{\mathsf{ElGm}}$ using the circuit compiler of [ide] over the scalar field of the BN254 curve. We present proof and proving key sizes after applying point compression. (Note that results depend heavily on machine specifics; e.g., Groth16 proof verification is reported [Bot] to be 100 times faster than our numbers using the same zk-SNARK 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.
Proving system	**Groth16** [Gro16] implementation of [ide]					
Constraints	4 226	8 830	8 404	1 564 556	1 716 794	219 740
Prov. key (pk) size	0.8 MB	2.15 MB	2.05 MB	530 MB	566 MB	62.5 MB
pk verif. time ($\approx$)	0.64 s	0.97 s	0.93 s	3 h 55 min	4 h 43 min	4 min 38 s
Verif. key size	1.75 kB	2.75 kB	2.1 kB	3.75 kB	3.6 kB	2.2 kB
Proving time	0.5 s	0.8 s	0.7 s	50 s	60 s	4.7 s
Proof size	402 B					
Proof verif. time	0.4 s					
Proving system	**Plonk** [GWC19] implementation of [Plo]					
Constraints	4 226	8 830	8 404	1 564 556	1 716 794	219 740
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
Proving time	1.2 s	1.5 s	1.7 s	2 m 50 s	2 m 53 s	33 s
Proof size	1.4 kB					
Proof verif. time	12 ms					

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

Our 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). Note that the outputs of Ψ_p are in $\mathbb{F}_p$, whereas α and β should be uniform in $\mathbb{F}_q$. For our choice of *Groth16* and *PlonK* parameters we have $p=(8-\epsilon)\cdot q$ with $\epsilon<2^{-124}$, and for our *Spartan* configuration we have $p=(1+\epsilon)\cdot q$ with $\epsilon<2^{-127}$. Therefore taking uniform values in $\mathbb{F}_p$ modulo q is statistically close to uniform values in $\mathbb{F}_q$, and (assuming security of DHIES with Poseidon) α and β modulo q are distributed (almost) as required.

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, namely BJB and Poseidon, resp., 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.[26]

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 $m = (\mathit{info}, m')$ with $\mathit{info} = \mathit{prd}$ will be signed, it suffices if the user encrypts m' instead of m in its first protocol message. This 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.

(B1) in Table 1 gives performance upper bounds for fully blind signature issuing, which deteriorate compared to scenario (A). An inherent reason is that secp256k1 and SHA-256 are not efficiently arithmetizable when using *Groth16* and *PlonK* over the BN254 curve. This is aggravated by the prototype implementation of the secp256k1 curve using simulated modulo reduce by [0xP].

Scenario (B2) showcases 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 $\mathsf{P}(\mathit{prd}, m, w)$ 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. Note that that this generalization has very small repercussions on efficiency compared to (B1).

Scenario (B3) is the same as (B2), but using the curve BJB instead of secp256k1. This could give a rough estimate for what performance could be achieved when *PlonK* and *Groth16* are instantiated over a pairing-friendly curve of appropriate order, for example following the approach by Sun et al. [SSS+22] combined with recently developed FFT techniques for non-smooth fields [BCKL21].

[26] Since elliptic-curve scalar multiplication in the BJB group for *fixed* base requires roughly 770 R1CS constraints as opposed to 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 2. Benchmarking scenario **(B2)** when instantiating NArg with *Spartan* [Set20] (which does not require a trusted setup) using a custom adaptation with efficient support for secp256k1 arithmetization by [TS], which is based on [Set] and [Bha]. The implementations are prototypes and currently do *not* make use of parallelization techniques and do not yet consider various optimizations for the secq256k1 curve used in the commitment scheme, and the secp256k1 curve used in the circuit.

Proving system	Constraints	*crs* size	Proving time	Proof size	Proof verif. time
Spartan [Set20, TS]	224 555	36 kB	2 m 34 s	36.6 kB	14 s

5.5 NIZKs with secp256k1 Support

The bulk of state-of-the-art NIZK implementations do not support efficient arithmetization for arbitrary choices of the Schnorr group $\mathbb{G}$, in particular not for the secp256k1 group. Non-native simulation of group operations creates significant overhead [0xP, EL], which we identify as the main source of inefficiency in scenarios (B1) and (B2) in Table 1. There are essentially three requirements that can restrict the possible choices of the arithmetization field of a NIZK: (i) The field must be large enough to bound the soundness error. (ii) The NIZK requires *explicit* properties of the field (e.g., having smooth multiplicative subgroups for FFT or FRI [BBHR18a] support). (iii) The NIZK requires *implicit* properties of the field (e.g., it uses a commitment scheme whose messages are elements of the field).

There now exist elegant ways around these requirements: (i) is commonly addressed by making use of extension fields and repeated proving [BBHR18b, Zer]. Ben-Sasson et al. [BCKL21] overcome (ii) by devising an FFT algorithm for arbitrary fields. To avoid (iii), there are generic solutions when the NIZK uses elliptic-curve-based commitments: one generates a new curve by using for example the Cocks-Pinch method to obtain pairing-friendly curves,[27] or complex multiplication for ordinary curves [BS08].

As the pairing-based schemes *Groth16* and *PlonK* are particularly subject to (ii) and (iii), we used *Spartan* as an alternative. It does not suffer from restriction (ii), as it does not require FFTs, and it alleviates (iii), as it requires ordinary rather than pairing-friendly curves, which are significantly easier to construct with lower overhead in the base field size. In the particular case of the secp256k1 base field, its size is large enough to satisfy (i), and a matching curve does not even need to be generated as it has the aforementioned "twin" curve secq256k1, whose order is equal to the field size of secp256k1 [SS11].

We follow Tehrani and Sankar [TS] and use this "twin" for *Spartan*'s commitment scheme to get a NIZK that is fine-tuned for scenario (B), i.e., blindly signing Bitcoin transactions. In Table 2 we give the results of testing scenario (B2) using this instantiation of *Spartan*. We remark that [TS] and the projects it builds upon [Set, Bha] are in an early development stage and therefore lack certain run-time optimizations. This is reflected by comparing the proving time in Table 2 to the one for (B2) in Table 1. However, due to the efficient support for secp256k1, the circuit complexity was reduced by more than a factor 7, which indicates the potential for further improvements in running time.

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 and the anonymous reviewers for EUROCRYPT'24 for their helpful comments.

[27] The drawback of using Cocks-Pinch is that the bit length of the base filed is up to twice as long [FST10], and, more importantly, it only yields the curve parameters, but no security guarantees, let alone efficient implementations.

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, 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*, pages 371–397. Springer, 2019.

[AF96] Masayuki Abe and Eiichiro Fujisaki. How to date blind signatures. In Kwangjo Kim and Tsutomu Matsumoto, editors, *ASIACRYPT'96*, volume 1163 of *LNCS*, pages 244–251. Springer, 1996.

[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*, pages 209–236. Springer, 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*, pages 191–219. Springer, 2016.

[AO00] Masayuki Abe and Tatsuaki Okamoto. Provably secure partially blind signatures. In Mihir Bellare, editor, *CRYPTO 2000*, volume 1880 of *LNCS*, pages 271–286. Springer, 2000.

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

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

[BBHR18a] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Fast Reed-Solomon interactive oracle proofs of proximity. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *ICALP 2018*, volume 107 of *LIPIcs*, pages 14:1–14:17. Schloss Dagstuhl, 2018.

[BBHR18b] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable, transparent, and post-quantum secure computational integrity. Cryptology ePrint Archive, Report 2018/046, 2018. https://eprint.iacr.org/2018/046.

[BC23] Benedikt Bünz and Binyi Chen. Protostar: Generic efficient accumulation/folding for special-sound protocols. In Jian Guo and Ron Steinfeld, editors, *ASIACRYPT 2023, Part II*, volume 14439 of *LNCS*, pages 77–110. Springer, 2023.

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

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

[BCI+10] Eric Brier, Jean-Sébastien Coron, Thomas Icart, David Madore, Hugues Randriam, and Mehdi Tibouchi. Efficient indifferentiable hashing into ordinary elliptic curves. In Tal Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 237–254. Springer, 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 Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part IV*, volume 13510 of *LNCS*, pages 517–550. Springer, 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*, pages 114–131. Springer, 2009.

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

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

[BCR+19] Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P. Ward. Aurora: Transparent succinct arguments for R1CS. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part I*, volume 11476 of *LNCS*, pages 103–128. Springer, 2019.

[BDL+12] Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. High-speed high-security signatures. *Journal of Cryptographic Engineering*, 2(2):77–89, 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*, pages 454–473. Springer, 2012.

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

[BF01] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the Weil pairing. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 213–229. Springer, 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*, pages 218–235. Springer, 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*, pages 121–151. Springer, 2020.

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

[BFP21] Balthazar Bauer, Georg Fuchsbauer, and Antoine Plouviez. The one-more discrete logarithm assumption in the generic group model. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part IV*, volume 13093 of *LNCS*, pages 587–617. Springer, 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*, pages 559–590. Springer, 2021.

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

[BFS20] Benedikt Bünz, Ben Fisch, and Alan Szepieniec. Transparent SNARKs from DARK compilers. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part I*, volume 12105 of *LNCS*, pages 677–706. Springer, 2020.

[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.

[Bha] Nalin Bhardwaj. Middleware to compile circom circuits to nova prover. Available at https://github.com/nalinbhardwaj/Nova-Scotia.

[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 Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part IV*, volume 13510 of *LNCS*, pages 551–581. Springer, 2022.

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

[BLL+21] Fabrice Benhamouda, Tancrède Lepoint, Julian Loss, Michele Orrù, and Mariana Raykova. On the (in)security of ROS. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part I*, volume 12696 of *LNCS*, pages 33–53. Springer, 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*, pages 514–532. Springer, 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*, pages 319–331. Springer, 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, 2003.

[Bol03] Alexandra Boldyreva. Threshold signatures, multisignatures and blind signatures based on the gap-Diffie-Hellman-group signature scheme. In Yvo Desmedt, editor, *PKC 2003*, volume 2567 of *LNCS*, pages 31–46. Springer, 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*, pages 62–73. ACM Press, 1993.

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

[BRS20] Samuel Brack, Leonie Reichert, and Björn Scheuermann. CAUDHT: decentralized contact tracing using a DHT and blind signatures. In Hwee-Pink Tan, Lyes Khoukhi, and Sharief Oteafy, editors, *Local Computer Networks LCN 2020*, pages 337–340. IEEE, 2020.

[BS08] Reinier Bröker and Peter Stevenhagen. Constructing elliptic curves of prime order. *Contemp. Math*, 463:17–28, 2008.

[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.

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

[CBBZ23] Binyi Chen, Benedikt Bünz, Dan Boneh, and Zhenfei Zhang. HyperPlonk: Plonk with linear-time prover and high-degree custom gates. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part II*, volume 14005 of *LNCS*, pages 499–530. Springer, 2023.

[CFN90] David Chaum, Amos Fiat, and Moni Naor. Untraceable electronic cash. In Shafi Goldwasser, editor, *CRYPTO'88*, volume 403 of *LNCS*, pages 319–327. Springer, 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*, pages 345–356. ACM Press, 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*, pages 229–243. ACM Press, 2017.

[Cha82] David Chaum. Blind signatures for untraceable payments. In David Chaum, Ronald L. Rivest, and Alan T. Sherman, editors, *CRYPTO'82*, pages 199–203. Plenum Press, New York, USA, 1982.

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

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

[CKM+23] Elizabeth C. Crites, Chelsea Komlo, Mary Maller, Stefano Tessaro, and Chenzhi Zhu. Snowblind: A threshold blind signature in pairing-free groups. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part I*, volume 14081 of *LNCS*, pages 710–742. Springer, 2023.

[CL01] Jan Camenisch and Anna Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In Birgit Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 93–118. Springer, 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*, pages 56–72. Springer, 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*, pages 229–235. Springer, 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*, pages 769–793. Springer, 2020.

[CP93] David Chaum and Torben P. Pedersen. Wallet databases with observers. In Ernest F. Brickell, editor, *CRYPTO'92*, volume 740 of *LNCS*, pages 89–105. Springer, 1993.

[DGS+18] Alex Davidson, Ian Goldberg, Nick Sullivan, George Tankersley, and Filippo Valsorda. Privacy pass: Bypassing internet challenges anonymously. *Proc. Priv. Enhancing Technol.*, 2018(3):164–180, 2018.

[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*, pages 309–324. Springer, 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*, pages 391–408. Springer, 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*, pages 233–253. Springer, 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*, pages 60–77. Springer, 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*, pages 33–62. Springer, 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*, pages 1651–1662. ACM Press, 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*, pages 519–534. Springer, 2017.

[FO18] Georg Fuchsbauer and Michele Orrù. Non-interactive zaps of knowledge. In Bart Preneel and Frederik Vercauteren, editors, *ACNS 18*, volume 10892 of *LNCS*, pages 44–62. Springer, 2018.

[FO22] Georg Fuchsbauer and Michele Orrù. Non-interactive Mimblewimble transactions, revisited. In Shweta Agrawal and Dongdai Lin, editors, *ASIACRYPT 2022, Part I*, volume 13791 of *LNCS*, pages 713–744. Springer, 2022.

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

[FOS19] Georg Fuchsbauer, Michele Orrù, and Yannick Seurin. Aggregate cash systems: A cryptographic investigation of Mimblewimble. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part I*, volume 11476 of *LNCS*, pages 657–689. Springer, 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*, pages 132–149. Springer, 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*, pages 63–95. Springer, 2020.

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

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

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

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

[FV10] Georg Fuchsbauer and Damien Vergnaud. Fair blind signatures without random oracles. In Daniel J. Bernstein and Tanja Lange, editors, *AFRICACRYPT 10*, volume 6055 of *LNCS*, pages 16–33. Springer, 2010.

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

[Gha17] Essam Ghadafi. Efficient round-optimal blind signatures in the standard model. In Aggelos Kiayias, editor, *FC 2017*, volume 10322 of *LNCS*, pages 455–473. Springer, 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*, pages 698–728. Springer, 2018.

[GKR+21] Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. Poseidon: A new hash function for zero-knowledge proof systems. In Michael Bailey and Rachel Greenstadt, editors, *USENIX Security 2021*, pages 519–535. USENIX Association, 2021.

[GO94] Oded Goldreich and Yair Oren. Definitions and properties of zero-knowledge proof systems. *Journal of Cryptology*, 7(1):1–32, 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*, pages 321–340. Springer, 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*, pages 305–326. Springer, 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*, pages 630–648. Springer, 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, 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*, pages 43–60. Springer, 2016.

[HEK12] Yan Huang, David Evans, and Jonathan Katz. Private set intersection: Are garbled circuits better than custom protocols? In *NDSS 2012*. The Internet Society, 2012.

[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.

[HK16] Lucjan Hanzlik and Kamil Kluczniak. A short paper on blind signatures from knowledge assumptions. In Jens Grossklags and Bart Preneel, editors, *FC 2016*, volume 9603 of *LNCS*, pages 535–543. Springer, 2016.

[HK23] Dominik Hartmann and Eike Kiltz. Limits in the provable security of ECDSA signatures. In Guy N. Rothblum and Hoeteck Wee, editors, *TCC 2023, Part IV*, volume 14372 of *LNCS*, pages 279–309. Springer, 2023.

[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*, pages 323–341. Springer, 2007.

[HKLN20] Eduard Hauck, Eike Kiltz, Julian Loss, and Ngoc Khanh Nguyen. Lattice-based blind signatures, revisited. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part II*, volume 12171 of *LNCS*, pages 500–529. Springer, 2020.

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

[HLW23] Lucjan Hanzlik, Julian Loss, and Benedikt Wagner. Rai-choo! Evolving blind signatures to the next level. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part V*, volume 14008 of *LNCS*, pages 753–783. Springer, 2023.

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

[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*, pages 150–164. Springer, 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*, pages 468–492. Springer, 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*, pages 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*, pages 98–127. Springer, 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*, pages 944–961. IEEE Computer Society Press, 2018.

[KPV22] Assimakis A. Kattis, Konstantin Panarin, and Alexander Vlasov. RedShift: Transparent SNARKs from list polynomial commitments. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022*, pages 1725–1737. ACM Press, 2022.

[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*, pages 49–62. Springer, 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*, pages 339–346. Springer, 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*, pages 20–35. Springer, 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. 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, 2002.

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

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

[PFM+22] Luke Pearson, Joshua Fitzgerald, Héctor Masip, Marta Bellés-Muñoz, and Jose Luis Muñoz-Tapia. PlonKup: Reconciling PlonK with plookup. Cryptology ePrint Archive, Report 2022/086, 2022. https://eprint.iacr.org/2022/086.

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

[Poe16] Andrew Poelstra. Mimblewimble, 2016. Available at https://download.wpsoftware.net/bitcoin/wizardry/mimblewimble.pdf.

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

[PS96] David Pointcheval and Jacques Stern. Security proofs for signature schemes. In Ueli M. Maurer, editor, *EUROCRYPT'96*, volume 1070 of *LNCS*, pages 387–398. Springer, 1996.

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

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

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

[Sch01] Claus-Peter Schnorr. Security of blind discrete log signatures against interactive attacks. In Sihan Qing, Tatsuaki Okamoto, and Jianying Zhou, editors, *ICICS 01*, volume 2229 of *LNCS*, pages 1–12. Springer, 2001.

[Set] Srinath Setty. Spartan: High-speed zkSNARKs without trusted setup. Available at https://github.com/microsoft/Spartan.

[Set20] Srinath Setty. Spartan: Efficient and general-purpose zkSNARKs without trusted setup. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part III*, volume 12172 of *LNCS*, pages 704–737. Springer, 2020.

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

[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. Cryptology ePrint Archive, Report 022/1079, 2022. https://eprint.iacr.org/2022/1079.

[TS] Daniel Tehrani and Lakshman Sankar. The fastest in-browser verification of ECDSA signatures in ZK, using Spartan on the secq256k1 curve. Available at https://github.com/personaelabs/spartan-ecdsa.

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

[Wag02] David Wagner. A generalized birthday problem. In Moti Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, pages 288–303. Springer, 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. https://tches.iacr.org/index.php/TCHES/article/view/8348.

[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*, pages 160–164. IEEE Computer Society Press, 1982.

[Zer] Polygon Zero. Plonky2: Fast recursive arguments with PLONK and FRI. 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

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

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

$\mathsf{wOMDL}^{\mathsf{A}}_{\mathsf{GrGen}}(1^\lambda)$	$\textsc{Chal}()$	$\textsc{DLog}(i)$
$(q, \mathbb{G}, G) \leftarrow \mathsf{GrGen}(1^\lambda)$	$x \leftarrow\$ \mathbb{Z}_q\,;\ X := xG$	$\vec{q} = \vec{q} \,\\|\, \vec{x}_i$
$\vec{x} := [\,]\,;\ \vec{q} := [\,]$	$\vec{x} = \vec{x} \\| x$	**return** $\vec{x}_i$
$y \leftarrow \mathsf{A}^{\textsc{Chal},\textsc{DLog}}(q, \mathbb{G}, G)$	**return** X	
return $\big(\lvert\vec{x}\rvert > 0 \wedge y \in \vec{x} \wedge y \notin \vec{q}\big)$		

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.*

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

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 $\textsc{Chal}()$, with probability $1 - P$, for some value P, it embeds Z in its response and aborts if A ever queries $\textsc{DLog}$ at a position at which Z was embedded:

$\underline{\mathsf{B}(q, \mathbb{G}, G, Z)}$

$\vec{x} \coloneqq [\ ]$ // empty list of tuples
$y \leftarrow \mathsf{A}^{\textsc{Chal},\textsc{DLog}}(q, \mathbb{G}, G)$
foreach (b, x) **in** $\vec{x}$:
  if $Z = (y - x)G$:
    return $y - x$
return 0

$\underline{\textsc{Chal}()}$

$\delta \leftarrow\!\!\$\ [0, 1]$
$x \leftarrow\!\!\$\ \mathbb{Z}_q\ ;\ X \coloneqq xG$
if $\delta > P$: $\vec{x} = \vec{x} \| (0, x)$
  return $Z + X$
$\vec{x} = \vec{x} \| (1, x)$
return X

$\underline{\textsc{DLog}(i)}$

if $\vec{x}_i[0] = 0$:
  abort
return $\vec{x}_i[1]$

Consider an adversary A against wOMDL that makes q $\textsc{DLog}$ 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 $\alpha(P_{max}) = \frac{1}{q}(1 - \frac{1}{q+1})^{q+1}$. $\square$

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 $\mathsf{PBS.Setup}$ hence is replaced by the setup from Figure 4. In $\textsc{Sign}_1$, the call $\mathsf{PBS.Sign}_1$ is instantiated by sampling $r \leftarrow\!\!\$\ \mathbb{Z}_q$ and returning $R = rG$, and in $\textsc{Sign}_2$, we instantiate $\mathsf{PBS.Sign}_2$ as defined in Figure 4, by a NIZK verification and return 0 if verification failed.

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

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

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

$(q, \mathbb{G}, G, \mathsf{H}) \leftarrow \mathsf{Sch.Setup}(1^\lambda)$
$(crs, \tau) \leftarrow \mathsf{NArg.Setup}(q, \mathbb{G}, G, \mathsf{H})$
$(ek, dk) \leftarrow \mathsf{PKE.KeyGen}(1^\lambda)$
$x \leftarrow\$ \mathbb{Z}_q \;\; ; \;\; X := xG$
$vk := (crs, ek, X)$
$\vec{\mathrm{S}} := [\,]; \; \mathrm{P\vec{R}D} := [\,]$
$\boxed{\vec{\mathrm{M}} := [\,]; \vec{\alpha} := [\,]; \vec{\beta} := [\,];}$
$\boxed{\vec{\mathrm{D}} := [\,]; \; \mathcal{Q} := \emptyset}$
$\big(m_i^*, \sigma_i^* := (R_i^*, s_i^*)\big)_{i \in [n]} \leftarrow \mathsf{A}^{\textsc{Sign}_1, \textsc{Sign}_2}(vk)$

if $\big(\exists\, i \in [n]\; \exists\, j \in [|\vec{\mathrm{S}}|] : m_i^* = \vec{\mathrm{M}}_j \wedge$
$\quad \vec{\mathrm{S}}_j \neq \varepsilon \wedge \vec{\mathrm{S}}_j[0] + \vec{\alpha}_j G + \vec{\beta}_j X = R_i^*\big)$:
return 0 **(II)**

return $\big(\; n > 0$
$\quad \wedge \forall\, i \in [n]\colon \mathsf{PBSch.Ver}(vk, m_i^*, \sigma_i^*) = 1$
$\quad \wedge \forall\, i \neq j \in [n]\colon (m_i^*, \sigma_i^*) \neq (m_j^*, \sigma_j^*)$
$\quad \wedge \nexists\, f \in \mathsf{InjF}([n], [|\mathrm{P\vec{R}D}|])\colon$
$\qquad \forall\, i \in [n]\colon \mathsf{P}(\mathrm{P\vec{R}D}_{f(i)}, m_i^*) = 1\big)$

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

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

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

if $\vec{\mathrm{S}}_j = \varepsilon$: **return** $\perp$
$(R, r, C, prd) := \vec{\mathrm{S}}_j$
$\theta := (X, R, c, C, prd, ek)$
if $\mathsf{NArg.Vfy}(crs, \theta, \pi) = 0$:
return 0
$\boxed{R' := R + \vec{\alpha}_j G + \vec{\beta}_j X}$
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)**
$\vec{\mathrm{S}}_j := \varepsilon; \; \mathrm{P\vec{R}D} = \mathrm{P\vec{R}D} \| prd$
$\boxed{\textbf{return } \vec{\mathrm{D}}_j}$
return $(r + cx) \bmod q$

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 $> i$.

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 $\mathsf{NArg.Rel}$, which is defined as $\mathsf{Sch.Setup}$. Reduction S, run in game SND therefore perfectly simulates G_1 to adversary A until abort. In $\textsc{Sign}_2$, it checks whether for a valid statement/proof pair (θ, π) parts of the supposed witness m, α and β satisfy $c \neq (\mathsf{H}(R', X, m) + \beta) \bmod q$ 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 $\textsc{Sign}_2$ 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

$\underline{\mathsf{S}(crs)}$

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

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

$r \leftarrow_\$ \mathbb{Z}_q\ ;\ R \coloneqq rG$

$\boxed{\begin{array}{l}(m, \alpha, \beta) \coloneqq \mathsf{PKE.Dec}(dk, C)\\ \vec{\mathrm{M}} = \vec{\mathrm{M}} \| m\ ;\ \vec{\alpha} = \vec{\alpha} \| \alpha\ ;\ \vec{\beta} = \vec{\beta} \| \beta\end{array}}$

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

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

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

$\boxed{\begin{array}{l}R' \coloneqq R + \vec{\alpha}_j G + \vec{\beta}_j X\\ \textbf{if } c \not\equiv_q \mathsf{H}(R', X, \vec{\mathrm{M}}_j) + \vec{\beta}_j \ \vee \mathsf{P}(prd, \vec{\mathrm{M}}_j) \neq 1 :\\ \quad \textbf{stop and return } (\theta, \pi) \quad \textbf{(I)}\end{array}}$

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

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

valid statement, meaning that there exists $w' \coloneqq (m', \alpha', \beta', \rho')$ s.t. $\mathsf{R_{Sch}}(par_\mathsf{R}, \theta, w') = 1$, which means:

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

where $R' \coloneqq R + \alpha' G + \beta' X$. 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

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

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

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

<u>Reduction to DL.</u> 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.

$\mathsf{L}^{\textsc{Chal},\textsc{DLog}}(q, \mathbb{G}, G)$

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

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

return $\perp$

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

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

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

if $\vec{\mathrm{S}}_j = \varepsilon$: **return** $\perp$
$(R, C, prd) := \vec{\mathrm{S}}_j$
$\theta := (X, R, c, C, prd, ek)$
if $\mathsf{NArg.Vfy}(crs, \theta, \pi) = 0$:
$\quad$ **return** $\perp$
$R' := R + \vec{\alpha}_j G + \vec{\beta}_j X$
if $c \not\equiv_q \mathsf{H}(R', X, \vec{\mathrm{M}}_j) + \vec{\beta}_j \ \vee \mathsf{P}(prd, \vec{\mathrm{M}}_j) \neq 1$:
$\quad$ **abort and return 0** **(I)**
$\vec{\mathrm{S}}_j := \varepsilon; \vec{\textsc{Prd}} = \vec{\textsc{Prd}} \| prd$
$r := \textsc{DLog}(j)$
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 $\textsc{Sign}_1$, L queries its challenge oracle $R \leftarrow \textsc{Chal}()$. Since the oracle returns uniformly sampled elements, the simulation is perfect up to this point. If A closes a session with session number j successfully with a call to $\textsc{Sign}_2$, L obtains $r \coloneqq \textsc{DLog}(j)$ from its oracle $\textsc{DLog}$.

Assume A satisfies condition E from (14). Thus some session number j with challenge $R_j \coloneqq \vec{\mathrm{S}}_j[0]$ was not closed, and so the oracle $r_j \coloneqq \textsc{DLog}(j)$ was not called either. Also for some index $i \in [n]$, we have $R_j + \vec{\alpha}_j G + \vec{\beta}_j X = R_i^*$ and by the assertion that A wins G_1, we know by the validity of the signatures that $s_i^* G = R_i^* + \mathsf{H}(R_i^*, X, m_i^*)X$. Combining these two equations yields $s_i^* G = r_j G + \vec{\alpha}_j G + \vec{\beta}_j x G + \mathsf{H}(R_i^*, X, m_i^*) x G$, 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 $r_j \coloneqq \log_G(R_j)$ and thereby wins game wOMDL, since r_j is the discrete logarithm of a challenge that was not solved by the oracle $\textsc{DLog}$, as required by the game.

Therefore we obtain:

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

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 $\textsc{DLog}$ oracle is also bounded by q, and by applying Lemma 1 we obtain an adversary D playing in the game DL where

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

$\underline{\mathsf{G}_3}$. 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 $sk \coloneqq (q, \mathbb{G}, G, \mathsf{H}, x)$ by running $(\bar{R}, \bar{s}) \leftarrow \mathsf{Sch.Sign}(sk, m)$. Next we replace the random signer challenge $R \coloneqq rG$ by $R \coloneqq \bar{R} - \alpha G - \beta X$. Note that $\mathsf{Sch.Sign}$ returns a uniform $\bar{R}$ and hence R is a uniform element and thus the simulation is perfect up to here. As a last change in $\textsc{Sign}_1$, we compute and append $(\bar{s} - \alpha) \bmod q$ to the list $\vec{\mathrm{D}}$. In $\textsc{Sign}_2$ instead of returning $s \coloneqq (r + cx) \bmod q$ we return the value we previously stored in $\vec{\mathrm{D}}$, that is $s \coloneqq \bar{s} - \alpha = \vec{\mathrm{D}}_j$.

The user now obtains simulated elements $(R, s) = (\bar{R} - \alpha G - \beta X, \bar{s} - \alpha)$. By definition of $\mathsf{Sch.Sign}$ we have $\bar{s}G = \bar{R} + \mathsf{H}(\bar{R}, X, m)X$, and by the assertion that line (I) was not reached, we have $c \equiv_q \mathsf{H}(R + \alpha G + \beta X, X, m) + \beta$. 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

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

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

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

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

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

$\mathsf{F}^{\text{SIGN}}(sp, X)$

$(crs, \tau) \leftarrow \mathsf{NArg.Setup}(sp)$
$(ek, dk) \leftarrow \mathsf{PKE.KeyGen}(1^\lambda)$
$vk \coloneqq (crs, ek, X)$
$\vec{\mathrm{S}} \coloneqq [\,];\ \vec{\mathrm{PRD}} \coloneqq [\,]$
$\vec{\mathrm{M}} \coloneqq [\,]; \vec{\alpha} \coloneqq [\,]; \vec{\beta} \coloneqq [\,];$
$\vec{\mathrm{D}} \coloneqq [\,]; \mathcal{Q} \coloneqq [\,]$
$\mathcal{F} \leftarrow \mathsf{A}^{\text{SIGN}_1, \text{SIGN}_2}(vk)$
$(m^*, \sigma^*) \leftarrow\!\!\$\ \mathcal{F} \setminus \mathcal{Q}$
return (m^*, σ^*)

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

$(m, \alpha, \beta) \coloneqq \mathsf{PKE.Dec}(dk, C)$
$\vec{\mathrm{M}} = \vec{\mathrm{M}} \| m;\ \vec{\alpha} = \vec{\alpha} \| \alpha;\ \vec{\beta} = \vec{\beta} \| \beta$
$(\bar{R}, \bar{s}) \leftarrow \text{SIGN}(m)$
$\mathcal{Q} = \mathcal{Q} \,\|\, \big(m, (\bar{R}, \bar{s})\big)$
$R \coloneqq \bar{R} - \alpha G - \beta X$
$\vec{\mathrm{D}} = \vec{\mathrm{D}} \,\|\, \big((\bar{s} - \alpha) \bmod q\big)$
$\vec{\mathrm{S}} = \vec{\mathrm{S}} \| (R, r, C, prd)$
return R

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

REDUCTION OF sEUF-CMA OF SCHNORR TO 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 $(sp, X) \coloneqq vk$ and has access to a signing oracle SIGN. With the Schnorr parameters sp it completes $\mathsf{PBSch.Setup}$ computing the common reference string crs for NArg and a key pair (ek, dk) for PKE. Moreover, F initializes a list $\mathcal{Q}$ used to store the message/signature pairs from its signing oracle 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 SIGN_1 query by A forwards the call to its signing oracle SIGN. The simulation is perfect.

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

We first show that there exists an element $(m_{i^*}^*, \sigma_{i^*}^*) \in \mathcal{F}$ that is not in $\mathcal{Q}_{\text{cls}}$. If we had $\mathcal{F} \subseteq \mathcal{Q}_{\text{cls}}$ then there would exist 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). $\square$

Remark 1. If we considered the weaker definition of unforgeability obtained by moving $\mathrm{P\vec{R}D} = \mathrm{P\vec{R}D} \| prd$ from $\textsc{Sign}_2$ to $\textsc{Sign}_1$, the predicates of all opened sessions are included in $\mathrm{P\vec{R}D}$.

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 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, $\mathsf{PBS.Setup}$, $\mathsf{PBS.User}_0$, $\mathsf{PBS.User}_1$ and $\mathsf{PBS.User}_2$ are replaced by the instantiations defined in Figure 4. The variables st_0 and st_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 st_i, but we renamed it since in $\textsc{User}_2$ it becomes part of $\sigma_{i\oplus b}$.

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

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

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

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

Together with the triangular inequality this yields:

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

$\mathsf{BLD}^{\mathsf{A},b}_{\mathsf{PBSch}}(\lambda)$, $\boxed{\mathsf{G}^{\mathsf{A},b}_1}$, $\boxed{\mathsf{G}^{\mathsf{A},b}_2}$

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

$\textsc{User}_0(i)$

if $sess_i \neq \texttt{init}$: **return** $\perp$
$sess_i = \texttt{open}$
$M := (m_{i \oplus b}, \alpha_i, \beta_i)$
// encrypt fixed vector instead:
$M := (\bar{m}, 0, 0)$ // $\bar{m}$ ... arbitrary fixed msg
$C_i := \mathsf{PKE.Enc}(ek, M; \rho_i)$
return C_i

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

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

$\textsc{User}_2(i, s)$

if $sess_i \neq \texttt{await}$: **return** $\perp$
$sess_i = \texttt{closed}$
if $sG = R_i + c_i X$:
  $\sigma_{i \oplus b} = \big(R'_{i \oplus b}\,,\ (s + \alpha_i) \bmod q\big)$
else : $\sigma_{i \oplus b} = \perp$
if $(sess_0 = sess_1 = \texttt{closed})$:
  if $(\sigma_0 = \perp \vee \sigma_1 = \perp)$:
    return $(\perp, \perp)$
  return (σ_0, σ_1)
return ε

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.

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

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

$$\mathsf{Adv}^{\mathsf{CPA}}_{\mathsf{PKE},\mathsf{C}_b} := \big|\Pr[\mathsf{CPA}^{\mathsf{C}_b,1}_{\mathsf{PKE}}] - \Pr[\mathsf{CPA}^{\mathsf{C}_b,0}_{\mathsf{PKE}}]\big| = \big|\Pr[\mathsf{G}_1^{\mathsf{A},b}] - \Pr[\mathsf{G}_2^{\mathsf{A},b}]\big|$$

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

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

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

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

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.

for $b \in \{0,1\}$. Together with the triangular inequality, this yields:

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

<u>Reducing G_2 to perfect blindness of "plain" Blind Schnorr.</u> 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,

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

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

$\textsc{User}_0(i)$

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

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.

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

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

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 $\mathsf{E}_1, \dots, \mathsf{E}_{i-1}$ to extract the messages $m_1, \dots, m_{i-1}$. 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.[28]
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.

Finally, we note that even if we replaced the encryption of the message by a hash of it (which would mean making random-oracle style extractability assumptions), the efficiency gains would be modest: Proving time (the most complex part) would not improve much, as the circuit size would decrease by one elliptic-curve scalar multiplication, but would still require the same number of invocations of the hash function. (A minor advantage would be the reduced communication complexity and verification time if the blindly signed message is very long.)

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].

[28] 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 $j = 1 \dots i-1$. Since we assumed E_j runs in time $t_{\mathsf{B},j}$, we have $t_{\mathsf{B},i} := t_{\mathsf{A},i} + \sum_{j=1}^{i-1} t_{\mathsf{B},j} = t_{\mathsf{A},i} + \sum_{j=1}^{i-1} 2^{i-j-1} t_{\mathsf{A},j}$.

Concretely, the relation generator for $\mathsf{R_{Sch}}'$ runs $(q, \mathbb{G}, G, \mathsf{H}) \leftarrow \mathsf{Sch.Setup}(1^\lambda)$ (as for $\mathsf{R_{Sch}}$), 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

$$LHS_i := e(\textstyle\sum_{j=1}^{d} a_{i,j} pk_j, pk') \stackrel{?}{=} RHS_i$$

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

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

whose computation requires $m \cdot d$ multiplications (and m additions) in $\mathbb{F}_p$ and a multiscalar multiplication (MSM)[29] 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.

[29] 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.