Efficient Proofs of Possession for Legacy Signatures

Anna P. Y. Woo
University of Michigan

Alex Ozdemir
Stanford University

Chad Sharp
University of Michigan

Thomas Pornin
NCC Group

Paul Grubbs
University of Michigan

Abstract

Digital signatures underpin identity, authenticity, and trust in modern computer systems. Cryptography research has shown that it is possible to prove possession of a valid message and signature for some public key, without revealing the message or signature. These *proofs of possession* work only for specially-designed signature schemes. Though these proofs of possession have many useful applications to improving security, privacy, and anonymity, they are not currently usable for widely deployed, legacy signature schemes—like RSA, ECDSA, and Ed25519. Unlocking practical proofs of possession for these legacy signature schemes requires closing a huge efficiency gap.

This work brings proofs of possession for legacy signature schemes very close to practicality. Our design strategy is to encode the signature's verification algorithm as a rank-one constraint system (R1CS), then use a zkSNARK to prove knowledge of a solution. To do this efficiently we (1) design and analyze a new zkSNARK called Dorian that supports randomized computations, (2) introduce several new techniques for encoding hashes, elliptic curve operations, and modular arithmetic, (3) give a new approach that allows performing the most expensive parts of ECDSA and Ed25519 verifications outside R1CS, and (4) generate a novel elliptic curve that allows expressing Ed25519 curve operations very efficiently. Our techniques reduce R1CS sizes by up to 200× and prover times by 3-22×. We can generate a 240-byte proof of possession of an RSA signature over a message the size of a typical TLS certificate—two kilobytes—in only three seconds.

1 Introduction

Digital signatures are the foundation of identity, authenticity, and trust in modern computer systems. On the web, several signatures are verified for every TLS connection; signatures also allow authenticating domain name records through DNSSEC. Signatures underpin user identity as well, through systems like OAuth and passkeys. Signatures secure software supply chains through code signing and are used to bootstrap trust in trusted execution environments.

An extraordinarily useful design pattern for all these applications of signatures is *proofs of possession* (PoPs): proving knowledge of a valid (but hidden) message and signature pair for some public key, and optionally proving the message satisfies some predicate. The security, privacy, and anonymity benefits of proofs of signature possession are huge: they can make credentials more private [32, 54], allow untraceable anonymous payments [17], help people browse the internet anonymously [96], make X.509 certificates more secure [45], build better blocklisting [95, 105], combat misinformation [55, 80], and more [27, 39, 46, 70].

Despite these applications, and interest from practitioners [106, 107], PoPs are rarely used in practice today. Cryptographers have built specialized signature schemes with very efficient PoPs in terms of prover/verifier time and proof size, such as BBS, CL, or PS signatures [30, 34, 77, 89]; since these signatures are rarely

used in practice, the usability of these PoPs for the applications described above is limited. Conversely, for signature schemes that are widely used in practice—in particular, RSA-PKCS1v1.5, ECDSA-P256, and Ed25519 [64, 66, 68], with the SHA-256 and SHA-512 [63] hash functions—the state of the art PoPs are much too inefficient for practical use. For example, in the seminal Cinderella work, proving possession of a single TLS certificate signed with RSA took 137 seconds [45].

Our results. This work dramatically improves the state of the art for PoPs for the three legacy signature schemes mentioned above. Ours are now the fastest known PoPs for these signatures. Relative to the prior state of the art, we improve:

- RSA-PKCS1v1.5 proving time by $>3\times$ (with similar proof size),
- ECDSA-P256-SHA256 proving time by $>7\times$ (with $\approx 10\times$ smaller proofs and a faster verifier), and
- Ed25519 proving time by $>7\times$ (with $\approx 10\times$ smaller proofs).

For the same problem size as the Cinderella proof of possession described before, we are about $75\times$ faster than their reported prover time. (Our numbers are on newer hardware, and the comparison is inexact in other ways; however, the Cinderella authors did not respond to our repeated requests to release the code, so an apples-to-apples comparison was unfortunately impossible.) Our techniques lead to reductions of up to $200\times$ in constraint costs (S2) over baselines, greatly reducing the storage overhead of metadata like structured reference strings. We also show our techniques reduce the size of ECDSA-based ring signatures by roughly $\approx 300\times$ over the state of the art [37, 54]. We describe our contributions in the technical overview below.

1.1 Technical Overview

We focus on the following concrete PoP setting: given a public key pk as public input, the prover shows they know a hidden message/signature pair (m, σ) such that $\mathsf{Vf}(\mathsf{pk}, m, \sigma) = 1$. We conservatively assume that both the message and signature must be *completely* hidden. (We discuss more general settings, such as partial message disclosure, below.)

To begin, it is important to understand the state of the art approach to PoPs for legacy signatures. It has two steps. First, express the function Vf as a rank-one constraint system (R1CS) (S2)—a generalization of arithmetic circuits—over a finite field $\mathbb{F}$. (A number of tools exist that compile programs in special domain-specific languages, such as Z# or Circom, to R1CS.) Second, to actually generate and verify PoPs, use a zero-knowledge proof (e.g., a zero-knowledge succinct argument of knowledge, or zkSNARK) to prove knowledge of a satisfying assignment (pk, m, σ) of the R1CS. The pk is the public part of the assignment, and (m, σ) are the witnesses. Further details are not yet important, but one important fact is that the atomic unit of "cost" for this approach is the number of R1CS constraints: more constraints means (in general) that proving and verifying is slower, and proofs are larger.

This approach for building PoPs, which we can call "R1CS+zkSNARK" for short, has a number of advantages. With some caveats, it is agnostic to the choice of zkSNARK. Thus, applications with varying performance requirements can choose the zkSNARK that fits their needs. (For example, backends with very small proofs are useful for blockchain applications, whereas applications like zero-knowledge middleboxes can tolerate somewhat larger proofs but must minimize prover time [114].) Another advantage is that it allows very complex policies to be expressed on messages: for example, to turn a legacy ID like a passport into an anonymous credential, one must prove that the message m satisfies formatting constraints [96]. This can be done by representing the format checks as a program in a high-level language and compiling it to R1CS.

The important drawback of this approach comes from the first step, namely expressing Vf in R1CS. The R1CS representation must express all the operations in Vf as additions and multiplications over a single finite field; most commonly this field is the integers modulo a large (> 256-bit) prime p that is a parameter of

the underlying zkSNARK. For legacy signature schemes, though, usually *none* of the operations actually occur in $\mathbb{F}_p$; furthermore, Vf contains operations in several different algebraic structures—bitwise operations and shifts, arithmetic modulo powers of two, arithmetic modulo large integers $q \neq p$, elliptic curve point arithmetic, and even padding and other string manipulations. Because R1CS is an NP-complete language, all these operation types *can* be emulated in $\mathbb{F}_p$, but this emulation is often highly inefficient: an operation that is nearly free on a standard CPU can translate into hundreds of constraints. The core reason why this is the case is that, in addition to constraints representing the operation itself, the emulation must include constraints on the inputs, outputs, and intermediate values of the operation, to guarantee the emulation is (logically) sound.

Example: R1CS for addition modulo 2^{32}. An instructive example of this phenomenon can be found when examining an R1CS constraint representation of $c = a + b \mod 2^{32}$. (This computation occurs 176 times in the compression function of SHA-256.) An R1CS compiler would translate this into the constraint $a + b = o2^{32} + c$ for $a, b, c, o \in \mathbb{F}_p$ (where o represents the overflow mod-2^{32}). However, and crucially, this constraint is a sound check *only if* a, b, c, o are in the subset of $\mathbb{F}_p$ in the range $[0, 2^{32} - 1]$. This is not guaranteed by the (cryptographic) soundness of the zkSNARK. Thus, without "range checking" a, b, c, o, the constraint gives only a single linear equation in four variables in $\mathbb{F}_p$, which has many solutions, most of which are not also correct modulo 2^{32}.

Thus, the R1CS compiler must also emit constraints that range check the four values a, b, c, o. The most natural way to perform this range check is by adding auxiliary variables that represent the bit decomposition of the four values, then checking that recombining the bits gives the claimed value. Each of these auxiliary variables, however, must be correctly constrained so that it can only take bit values. This costs one constraint per bit; thus, a single addition modulo 2^{32} takes 129 R1CS constraints to represent. This means that representing the 176 of these in each SHA-256 compression costs roughly twenty-two thousand constraints. (This is of course an overestimate—optimizations are possible—but they only give small improvements.)

Similar issues arise with emulating the other operations performed in Vf for legacy signatures. A curious, but significant, fact is that in state of the art approaches the cost of range checking auxiliary variables typically dominates all other costs: for example, the state of the art R1CS for modular exponentiation is nearly 95% range-checking (S5).

Our approach. Despite the efficiency challenges of the R1CS+zkSNARK approach, our PoP protocols use it. We believe this choice is justified given our focus on building tools that different applications can be built on. Making usable tools for different applications requires flexibility, and the R1CS+zkSNARK approach has unparalleled flexibility. Thus, the main technical challenge we tackle in this work is improving the efficiency of the R1CS+zkSNARK approach. At a high level, we leverage two ideas to do this. First, we *reduce emulation overhead* through careful engineering and the application of lookup arguments, a new class of technical tool in proof systems. We also eliminate emulation entirely in some cases by customizing the backend zkSNARK algorithms to be "compatible" with the arithmetic of legacy signature schemes. The second main idea is to avoid emulating expensive operations (e.g., elliptic curve scalar multiplications) in R1CS, by moving the relevant part of the signature verification into a separate *sidecar protocol*.

Reducing emulation overhead. As discussed, emulation overhead comes mainly from checking the validity of intermediate values, and in particular checking they are in the expected range. To address this we apply a new technical tool: *lookup arguments*. Briefly, a lookup argument is an efficient way to express in R1CS lookups in fixed, public tables. More formally, the lookup verifies that a sequence of values $s_1, \ldots, s_k$ in $\mathbb{F}_p$ is in a public list T. Lookup arguments can naturally represent range checks by taking the list T as all the elements in the right range, and the s_is to be the elements to be range checked [25]. State of the art lookup arguments cost as little as one R1CS constraint per s_i, which seems to already yield huge improvements to range checking: for the mod-2^{32} addition example above, replacing bit decomposition with a lookup argument would reduce the cost from 32 constraints to just one.

Actually using lookup arguments to reduce emulation overhead is nontrivial for a number of reasons. First, lookup arguments require a zkSNARK with a new computational model. Most zkSNARKs support R1CS or other deterministic, circuit-like models. Lookups require either a model with explicit lookup constraints [87] or a model that supports interactive, randomized circuits [72] (in the language of computational complexity: the class MA) which can be used to simulate a lookup protocol. We adopt the latter approach: we use a zkSNARK for *Interactive R1CS* [83]. But, the only existing zkSNARKs for I-R1CS are Mirage [72] and Mirage+ [83], which have very small proofs, but larger proving time and a larger SRS. No existing zkSNARK offers better proving time and SRS size while also supporting I-R1CS.

Second, lookup arguments are only a net improvement if the high setup cost (linear in $|T|$) can be amortized. This has a couple of implications: first, it is impossible to naively apply lookup arguments to (say) the 32-bit addition problem above: the table setup would cost billions of constraints. Thus, one must split the problem into smaller tables that can be re-used. However, this then requires careful computation-specific planning of table sizes so smaller tables can be re-used. At the level of software, no approach to this kind of planning exists, or has even ever been articulated or even formalized.

Third, and most subtly, replacing bit decomposition with lookups changes the performance profile of the backend zkSNARK in a complex way. With current zkSNARK software, it is cheaper to prove constraints involving very small witnesses, due to input-dependent and non-constant-time optimizations for these cases. Current state of the art lookup arguments replace many constraints with small (one-bit) witnesses with a few constraints that have large witnesses. (It is not clear this is an inherent property of lookup arguments, but it is true for all known protocols.) Thus, it is not clear that the large reduction in constraint costs that comes from employing lookups will necessarily lead to a reduction in prover and verifier time.

We make a number of contributions to the three issues discussed above. First, in Section 3 we design and analyze a new zkSNARK that supports I-R1CS and can therefore prove statements that use lookup arguments. Our new scheme, Dorian, is based on the popular Spartan zkSNARK [98] and has a similar performance profile, which makes it attractive for applications like zero-knowledge middleboxes that can tolerate somewhat larger proofs. Importantly, like Spartan, Dorian is agnostic to the underlying elliptic curve group, which means (unlike Mirage) it can use customized curves. Intuitively, Dorian works by splitting the witness commitment into several phases, to allow the verifier to send random challenges that are used to derive subsequent witnesses. (This is of course compiled using Fiat-Shamir, with verifier challenges derived as the hashes of partial witness commitments, but it is easier to understand in the interactive setting.)

We ensure our use of lookup arguments is concretely efficient using different strategies for different parts of the Vf computation. For improving the R1CS representations of SHA256 and SHA512 (S4), we use them to enhance a known encoding of bitwise operations using constraints, due to [25]. Rather than using lookups to directly check results of bitwise operations on intermediate words of the state, that work proposes using $\mathbb{F}_p$ arithmetic to compute the bitwise operations, but on a "sparse" form of the word. Sparse form is better for bitwise operations but worse for mod-2^{32} additions, which can be done fairly cheaply in the normal "dense" form. We observe that lookup tables can be used to convert between sparse and dense form very cheaply; thus, we can ensure that the better of the two forms of each word of the state is always available. We also choose our limb representation of the 32-bit state words in an optimized way to both reuse tables and get some bitwise rotations for free. Our final R1CS for SHA-256 is only around six thousand constraints per block, plus a one-time setup.

For big-integer arithmetic mod $q \neq p$, as described above, we use lookup arguments to drastically reduce the cost of range-checking the intermediate limbs of a computation. In Section 5 we describe the basic approach for RSA; our techniques generalize to ECDSA and EdDSA as well. An interesting subtlety is that we must also range-check the carry values of limb-wise arithmetic; for efficiency we want to allow these to slightly exceed the limb width—doing so minimizes the number of reductions mod q we must express in R1CS—but this means we must efficiently use a fixed set of tables to range-check limbs of several different widths. We observe for the first time that the choice of limb widths, carry widths, and table sizes can be seen

as a compiler optimization problem. In Section 5.2, we give the first formalization of this as the *batched flexible range check* problem. We designed a planner to choose optimal sizes for our problems of interest.

In addition to these uses of lookup arguments, we apply several known ideas for accelerating elliptic curve operations, such as preprocessing multiples of known points and (safely) using incomplete formulas (S6). Finally, we extend the known idea of eliminating emulation entirely by changing the field of the underlying zkSNARK (S7). This renders our techniques less backend-agnostic, but produces huge performance improvements (even compared to our other best techniques). Notably, we generate a new elliptic curve that can natively prove ed25519 field operations, which we call T-25519. We run our Dorian proof system on top of this curve.

Sidecar protocols. In aggregate, our techniques for reducing emulation overhead are hugely impactful: we reduce constraint costs by two orders of magnitude or more in some cases. However, we still have some extremely expensive operations left in R1CS. The main culprit is the non-fixed-base elliptic curve scalar multiplication that is necessary to verify both ECDSA and Ed25519 signatures. Roughly, both signature verifications involve checking a relation like $\mathsf{pk} = aG + bR$ where a, b, R are two scalars and an elliptic curve point that are part of the signature, G is a fixed elliptic curve point, and pk is the signer's public key. In Section 6.1, we explain how we move this part of the verification outside of R1CS and into a separate protocol run alongside the "main" zkSNARK (hence sidecar). The main advantage of doing this is that the sidecar protocol works over group elements in the signature's native curve group; thus, statements about group elements can be checked directly rather than via emulation in the zkSNARK.

Instantiating our sidecar approach required tackling several challenges. First, it is not entirely straightforward to split the Vf operations between the sidecar protocol and the zkSNARK, especially for ECDSA where a non-algebraic truncation is applied to the point R. Second, for soundness is is necessary to securely "link" the common witnesses in the sidecar protocol and the main zkSNARK, because (e.g.) in an ECDSA PoP the scalar a must depend on the hash of the message, which is recomputed only in the zkSNARK. The final challenge is privacy: our proof of possession requires hiding the entire message and signature, so our sidecar protocol cannot reveal parts of the signature.

The starting point for our sidecar protocol is Okamoto's classic sigma protocol for proving knowledge of a representation. In our setting, Okamoto's protocol lets us prove we know a representation (a, b) of pk with respect to bases G and R. We can link this with the zkSNARK using an idea due to Sigmabus: we modify the sigma protocol so that the prover commits to the randomness and witnesses, then in the zkSNARK we prove the responses are correctly computed. This correctness proof is only a few modular multiplications—not elliptic-curve operations—and so is very cheap. This still leaves privacy: Okamoto's protocol only works if the verifier knows both bases G, R, but we cannot reveal R. We fix this by designing an extension of Okamoto's protocol to the case where one of the two bases is committed using Elgamal encryption. Because this encryption preserves the group structure, we can still verify the relevant algebraic relation holds. This transform re-introduces an elliptic curve operation into the zkSNARK; however, this is a fixed-base operation (not variable-base) and so is $\approx 4\times$ cheaper. We present the sidecar protocol for ECDSA, but a very similar idea also works for Ed25519.

Though it is not the main focus of this work, in Section 6.2 we show that our sidecar protocol can be adapted to a related setting: ECDSA-based ring signatures. Prior work [37, 54, 61] has shown how to reduce ring signature to proofs of two statements: first, prove knowledge of a valid signature for a public message with respect to a committed public key; second, prove that the committed key is on a list of keys (without revealing where on the list it is). We show that with a few tweaks, our sidecar protocol can prove the first statement very cheaply; we also show it is compatible with the Groth-Kohlweiss protocol for the second.

Results. We discuss our implementation and results in Section 8. All our code is open-source and permissively licensed, and available at https://github.com/pag-crypto/sigpop. We implemented all R1CS using the Z# language with the CirC compiler toolchain [82]. We implemented all other protocols in Rust.

We benchmark with Mirage and our Dorian zkSNARK.

Our improvements for hashing over the state of the art [25] were fairly modest—only about a 30% decrease in constraint size per block. Compared to the Z# standard library R1CS for SHA-256, our improvements were more substantial; roughly an 80% reduction in constraint sizes. With Mirage we can prove knowledge of a 2048-byte hash preimage in 2.6 seconds.

For RSA-PKCS1v1.5, an apples-to-apples comparison to prior academic state of the art gives $\approx 3\times$ faster prover time. Compared to the most popular open-source Circom implementation we found, we have a $\approx 22\times$ faster prover with comparable verifier times and proof sizes. Notably, our implementation has fairly practical prover times even for messages the size of TLS certificates; a qualitative takeaway of this is that hitherto-impractical applications [45] of ZKPs to the web PKI are now within reach.

For ECDSA P-256 and Ed25519, we benchmarked two parameter settings: one with right-field arithmetic (via the T-256 custom curve [54]) using our new Dorian zkSNARK, and one with our sidecar protocol and other optimizations (but without right-field) using Mirage. For ECDSA, in apples-to-apples comparisons both provers are roughly $10\times$ faster than the best open-source code we found. For Ed25519 in the same setting, we improve state-of-the-art by between roughly 8–$22\times$. Both provers achieve sub-three-second proving time even for a two-kilobyte message. Though the Dorian prover is somewhat faster than Mirage, we found that in both cases the cost of the algebraic part of ECDSA verification was so low that the prover costs are dominated by hashing; thus the improvement of Dorian is fairly small. Still, we expect Dorian will be useful in settings where potentially many elliptic curve operations must be proven in one circuit, or settings where the relatively larger size of Mirage's SRS is a consideration. For ECDSA-based ring signatures, we achieve an estimated $> 200\times$ reduction in signature size vs. the state of the art; the full implementation is a work in progress.

Extensions and discussion. In aggregate, our results represent a huge improvement in the state of the art for PoPs of legacy signature schemes. The most important qualitative takeaway of our work is that for PoPs or related applications that involve ZKPs of signatures, *large-integer algebra is no longer a performance bottleneck*—in all our experiments, the modular arithmetic and elliptic-curve operations that dominate costs on normal CPUs are now a low-order cost for proving.

In this work we mostly focused on optimizing the core proof protocols themselves; there are more interesting questions to answer in using our PoPs in applications. One question we did not study is integrating our PoPs with validity policies on messages. These policies are necessary in, e.g., converting a legacy credential to an anonymous credential [45]. We believe a feature of our approach is that the flexibility of R1CS makes it easy for application designers to layer message policies on top of our PoPs.

Our techniques are applicable to the most widely-used signature schemes today, but with the transition to post-quantum cryptography it is interesting to think about PoPs for the new NIST post-quantum signature standards. Achieving efficient protocols for these would likely require very different technical ideas, since the underlying algorithms are very different.

Finally, we do not explore the question of transferring our techniques to other computational models, such as the popular Plonkish language [58]. We suspect our techniques will lead to a big improvement in Plonkish and other encodings of legacy signature verifications, but leave the details to future work.

2 Preliminaries

Throughout, we will use standard cryptographic tools, such as commitment schemes and zero-knowledge proofs, and refer to their standard security properties without explicit definition. Readers needing more background should consult a textbook on cryptography [22, 69] or proof systems [104].

We use standard notation for sets of bitstrings (e.g., $\{0,1\}^*$) groups (G generates $\mathbb{G}$) and fields ($\mathbb{F}_p$ has order p). We use additive notation for groups. We use bracket notation to denote ranges: $[n] = \{0, \ldots, n-1\}$—sometimes we will abuse notation and instead use $[n]$ to refer $\{1, \ldots, n\}$—and we use the notation $|\cdot|$ to

denote the size of a string or integer in bits.

Cryptographic primitives. A signature scheme $\Sigma = (\mathsf{Kg}, \mathsf{Sign}, \mathsf{Vf})$ is a triple of algorithms. We omit explicit definitions of Kg and Sign. The algorithm Vf takes as input a public key pk, message m, and signature σ and outputs a bit b (where $b = 1$ means verification succeeded).

A language $\mathcal{L}$ is a subset of $\{0,1\}^*$. A *relation* $\mathcal{R}$ is a subset of $\{0,1\}^* \times \{0,1\}^*$. (Below we will sometimes consider relations with public parameters; for notational convenience we will implicitly include these parameters in inputs.) For an input x, we say that $x \in \mathcal{L}$ if $\exists w$ such that $(x, w) \in \mathcal{R}$.

Rank-one constraint systems (R1CS). Our starting point is R1CS, a popular NP-complete language that generalizes arithmetic circuit satisfiability.

Definition 1 (Rank-1 Constraint System)**.** *A Rank-1 Constraint System (R1CS) is a tuple* $(\mathbb{F}; m, n \in \mathbb{N}; A, B, C \in \mathbb{F}^{n \times m})$ *where* $m \geq \ell$*. For* $x \in \mathbb{F}^{\ell}$*, a vector* $w \in \mathbb{F}^{m-\ell}$ *is called an R1CS witness if for* $z = (x, w)$*,* $(A \cdot z) \circ (B \cdot z) = (C \cdot z)$ *where* $\circ$ *denotes element-wise multiplication.*

An important fact about R1CS is that linear operations (additions and multiplications by constants) on elements of z are "free"—they do not increase the problem size.

Interactive R1CS. Our new zkSNARK, Dorian, handles a generalization of R1CS for *interactive* proofs. This I-R1CS notion was defined in [83].

Definition 2 (I-R1CS)**.** *An I-R1CS is a tuple* $\mathcal{I} = (\mathbb{F}; \rho, m, n, \ell_x, \ell_{r_1}, \cdots, \ell_{r_\rho}, \ell_{w_1}, \cdots, \ell_{w_\rho} \in \mathbb{N}; A, B, C \in \mathbb{F}^{n \times m})$*. It defines a* ρ*-round proof system over* $\mathbb{F}$*, where in round* $i \in [\rho]$*,* P *sends a message* $w_i \in \mathbb{F}^{\ell_{w_i}}$ *and receives a uniformly random response* $r_i \in \mathbb{F}^{\ell_{r_i}}$*. The last response must be empty:* $\ell_{r_\rho} = 0$*.* V*'s test is defined by three matrices* $A, B, C \in \mathbb{F}^{n \times m}$ *where* n *is the number of constraints and* $m = \ell_x + \sum_i (\ell_{w_i} + \ell_{r_i})$ *is the number of variables.* V *accepts if* $(A \cdot z) \circ (B \cdot z) = (C \cdot z)$ *where* $z = (x, r_1, \cdots, r_\rho, w_1, \cdots, w_\rho)$*.*

An I-R1CS $\mathcal{I}$ is an arithmetization of an *interactive argument* between a prover P and verifier V (we define interactive argument in App. A.1) . Thus, all properties that are defined for an interactive proof are also defined for an I-R1CS. For example, $\mathcal{I}$ can be *special sound* for a relation $\mathcal{R}(x, w)$ (Def. 11; App. A.1) .

Lookup arguments. A *lookup* is an interactive proof to show that a sequence of elements $X = (x^{(1)}, \ldots, x^{(A)})$ are in a set (or *table*) S. Haböck gives a very efficient lookup for $S = \{s^{(1)}, \ldots, s^{(N)}\} \subset (\mathbb{F}_p)^k$ [62]. In this argument, P sends counts $c_1, \ldots, c_N \in [A+1] \subset \mathbb{F}_p$, where $c^{(i)}$ is the number of occurrences of $s^{(i)}$ in X. Then, V samples $\alpha, \beta \in \mathbb{F}_p$ and asserts that:

$$\sum_{i=1}^{N} \frac{c^{(i)}}{\alpha + \sum_{j=1}^{k} \beta^{j-1} s_j^{(i)}} = \sum_{i=1}^{A} \frac{1}{\alpha + \sum_{j=1}^{k} \beta^{j-1} x_j^{(i)}}$$

This protocol is complete and special-sound. When encoded in I-R1CS, it requires $k(N + A) + O(1)$ constraints. We will use instances of this protocol in our I-R1CS instances.

3 Dorian: Spartan with Verifier Randomness

Below we will make heavy use of tools like lookup arguments that require verifier randomness. To use these tools, the proof system must support I-R1CS. Prior work constructed Mirage [72] (an extension of Groth16), which implicitly supports 2-round I-R1CS. We use Mirage in part of our evaluation, but for technical reasons it is incompatible with the "right-field arithmetic" technique we describe in §7.

In this section, we give Dorian, a generalization of the Spartan [98] proof system to 2-round I-R1CS that is compatible with right-field arithmetic. As far as we know, Dorian is the first IOP-based proof system with

$\underline{\text{Dorian}_{\mathcal{I}} = \langle P(x, \hat{w}_0), V(x)\rangle}$:

$P : (C_0, S_0) \leftarrow\!\!\$\ \mathsf{PC.Commit}(\mathbf{pp}, \widetilde{\hat{w}_0})$ and send C_0 to V

$V : r_1 \leftarrow\!\!\$\ \mathbb{F}^{\ell_{r_1}}$ and send r_1 to P

$P : w_1 \leftarrow \mathcal{I}.\mathsf{P}(x, w_0, r_1)$; $(C_1, S_1) \leftarrow\!\!\$\ \mathsf{PC.Commit}(\mathbf{pp}, \widetilde{\hat{w}_1})$ and send C_1 to V

P and V define $x' = (x, r_1)$, P defines $w = \sum_i v_i \cdot \hat{w}_i$

P and V complete the Spartan protocol with x', w except that C_w is computed differently: Instead of engaging in $\mathsf{PC}_{\mathsf{Multi}}.\mathsf{Open}$ for $\tilde{w}(r_y[1:]) \to e_i$ to get C_w, both parties engage in $\mathsf{PC}_{\mathsf{Multi}}.\mathsf{Open}$ for $\tilde{w}_i(r_y[1:]) \leftarrow e_i$ for all $i = \{0, 1\}$ to get evaluation commitments C_{e_0} and C_{e_1}. The commitment to witness evaluation is then defined as $C_{v_w} = (\tilde{v}_0(r_y[1:]))C_{e_0} + (\tilde{v}_1(r_y[1:]))C_{e_1}$, and the final evaluation check is otherwise unchanged.

Figure 1: Dorian, our extension of Spartan that supports verifier randomness. All variables are as defined in the text.

the latter property. Spartan is a convenient starting point for us because it works for any group and has a fast prover and verifier.

The Spartan protocol. Spartan [98] is an interactive protocol for proving R1CS satisfiability. It interprets the R1CS matrices A, B, C as functions $\{0,1\}^\mu \times \{0,1\}^\mu \to \mathbb{F}$, and similarly $Z : \{0,1\}^\mu \to \mathbb{F}$, by writing the indices as their binary representations. Their multilinear extensions $\widetilde{A}, \widetilde{B}, \widetilde{C}, \widetilde{Z}$ defines a polynomial $\widetilde{F}_x$, which vanishes on $\{0,1\}^\mu$ if and only if the R1CS instance is satisfied. The R1CS satisfiability is then reduced to a claim that $\sum_{y \in \{0,1\}^\mu} \widetilde{F}_x(X) \cdot \widetilde{eq}(X, \tau) = 0$ for a random $\tau \in \{0,1\}^\mu$ provided by the verifier. The prover and verifier engage in a protocol for this claim.

One subprotocol requires the verifier to evaluate $\widetilde{F}_x$ at a random point, but the verifier cannot do this themselves since this polynomial reveals $\widetilde{Z}$ constructed from the secret witness w. To solve this, Spartan requires the prover to provide a commitment to the multilinear extension $\widetilde{w}$ of the witness at the beginning. Later, the prover sends to the verifier a commitment to the evaluation of $\widetilde{w}$ and an opening proof to show that the committed evaluation is correct with respect to the commitment at the beginning. Spartan can be made non-interactive using Fiat-Shamir.

Dorian: Spartan for interactive R1CS. Our Spartan extension Dorian supports 2-round I-R1CS (S2). Define the I-R1CS being proved as $\mathcal{I} = (\mathbb{F}; \rho, m, n, \ell_x, \ell_{r_1}, \ell_{w_0}, \ell_{w_1}, A, B, C)$ and the I-R1CS prover as $\mathcal{I}.\mathsf{P}$. The Z vector is (x, w_0, r_1, w_1), where $w_0 \in \mathbb{F}^{\ell_{w_0}}$ is P's first message, $r_1 \in \mathbb{F}^{\ell_{r_1}}$ is V's random challenge, and $w_1 \in \mathbb{F}^{\ell_{w_1}}$ is P's second message (which may depend on r_1). Roughly, Dorian works by representing w_0 and w_1 with *distinct* multilinear extensions, using auxiliary polynomials to fuse them together in the rest of the protocol.

More formally, let J_i be the indices of w_i in Z. Let $\hat{w}_i : \{0,1\}^{\mu'} \to \mathbb{F}$ agree with w_i on J_i, and be zero otherwise. Let $\mu' = \log |J_i|$. We can express $w : \{0,1\}^{\mu'} \to \mathbb{F}$ in terms of the $\hat{w}_i$. To achieve this, we express J_i as a multilinear polynomial $v_i : \{0,1\}^{\mu'} \to \{0,1\}$, where $v_i(x) = 1$ when $x \in J_i$ and 0 otherwise. Then, we have

$$w(y[1..]) = \sum_{i=0}^{1} v_i(y[1..]) \cdot \hat{w}_i(y[1..]).$$

During the protocol, the prover commits to $\hat{w}_0$ and $\hat{w}_1$ in rounds. Then, both parties use $w(\cdot) = \sum_i v_i(\cdot) \cdot \hat{w}_i(\cdot)$ as the witness for the rest of the protocol. The complete protocol is shown in Figure 1.

Theorem 1. *Let $\mathcal{I}$ be an I-R1CS that is complete and special sound for witness relation $\mathcal{R}$. Let $\mathsf{PC}_{\mathsf{Multi}}$ be complete and $(\sqrt{m}, (4_\pm, \ldots, 4_\pm), 2)$ special-sound. Then Dorian$_I$ with $\mathsf{PC}_{\mathsf{Multi}}$ is a zero-knowledge proof of knowledge for $\mathcal{R}$.*

Proof. The completeness follows from the completeness of the sum-check protocol, of the underlying polynomial commitment scheme and of the I-R1CS $\mathcal{I}$.

Zero-knowledge (ZK) can be proven by adapting the security proof from prior work [42]. Our ZK simulator Sim' is similar to theirs, with the following differences: In step 1, Sim' samples two random multilinear polynomials $\tilde{w}_0, \tilde{w}_1 \leftarrow_{\$} \mathbb{F}[X_1, \cdots, X_\mu]$, compute commitments $C_{\tilde{w}_i} \leftarrow \mathsf{PC}_{\mathsf{Multi}}.\mathsf{Commit}(\mathsf{pp}, \tilde{w}_i, \omega_{\tilde{w}_i})$ for $i \in \{0,1\}$, and append $(C_{\tilde{w}_i})_{i\in\{0,1\}}$ to the transcript tr. In step 10, Sim' generates two opening proofs for $C_{\tilde{w}_0}$ and $C_{\tilde{w}_1}$, obtaining the commitments $C_{v_{w_0}}, C_{v_{w_1}}$, respectively, where $v_{w_i} \leftarrow \tilde{w}_i(r_y[1\cdots])$ for $i \in \{0,1\}$. These two proofs are appended to the transcript tr. Then, Sim' computes $v_w = \sum_{i\in\{0,1\}} \tilde{v}_i(r_y[1\cdots])v_{w_i}$ and $C_w = \Pi_{i\in\{0,1\}} C_{v_{w_i}}^{\tilde{v}_i(r_y[1\cdots])}$, which are used in step 11. From the construction of Sim', the proofs produced are accepting, and the output is indistinguishable from that of real transcripts, thereby concluding the proof of ZK.

The knowledge soundness of the Fiat-Shamir compiled $\text{Dorian}_{\text{FS}}$ is satisfied if its interactive version Dorian is special sound (SS), according to the theorem of [42]. The SS property of Dorian will be proven in Lemma 1. □

Lemma 1. *Let $\mathcal{I}$ be an I-R1CS that is $\mathbf{n}$-special sound for witness relation $\mathcal{R}$. Then, Dorian_I for $\mathcal{R}$ is $\bar{\mathbf{n}}$-special sound, where $\bar{\mathbf{n}}$ is*

$$(\mathbf{n}, 1, (1, 2_{\mathsf{li}}, 2)^\mu, 2, 2, 2, 1, (1, 2_{\mathsf{li}}, 2)^\mu, (\underbrace{4_{\pm}, \cdots, 4_{\pm}}_{\mu/2}, 2)^2, 2).$$

Proof. We present a PPT tree extractor $\mathcal{TE}$ that, given a public statement x and a $\bar{\mathbf{n}}$-tree of accepting transcripts $\mathcal{T}$ from an EPT adversary $\mathcal{A}$, can extract w such that $(x, w) \in \mathcal{R}$. The tree extractor $\mathcal{TE}$ works as follows:

For each $\mathbf{n}'$ sub-tree in $\mathcal{T}$, it uses a PPT tree extractor $\mathcal{TE}'$ to extract witnesses w_0, w_1 such that (x, r_1, w_0, w_1) is valid for the I-R1CS $\mathcal{I}$, corresponding to the verifier's challenge r_1 labelled on the incoming edge of the sub-tree. If w_0 is the same for all $\mathbf{n}'$ sub-tree, the extracted witnesses form an accepting tree of witnesses $\mathcal{T}_w$ for $\mathcal{I}$. The tree extractor $\mathcal{TE}$ then uses a SS extractor for $\mathcal{I}$ to extract w from $\mathcal{T}_w$.

If $\mathcal{TE}'$ extracts valid witnesses for all $\mathbf{n}'$ sub-trees, then they form an accepting tree of witnesses $\mathcal{T}_w$ for $\mathcal{I}$ with overwhelming probability, due to the binding property of the underlying polynomial commitment scheme.

The construction of $\mathcal{TE}'$ is similar to the tree extractor from the SS proof [42] for Spartan except that in step 4 , the sub-extractor for the opening argument is run twice to extract multilinear polynomials $\tilde{w}_0, \tilde{w}_1$. As in [42], we consider two hybrid worlds, with the additional condition that we reject if the extracted R1CS witness is not satisfying, which happens with probability at most $\frac{7\mu+1}{\mathbb{F}}$. □

4 SHA2 with lookups

The SHA-256 hash function [63] operates on 32-bit words and performs both arithmetic and bitwise operations. The challenge when representing SHA-256 in a zkSNARK is to encode both operation types efficiently. If a 32-bit word w is represented by a field element with the same unsigned value as w, the arithmetic operations (the additions) are somewhat cheap. But then, the bitwise operations are expensive.

For example, consider the AND of $u, w \in \{0, \ldots, 2^{32} - 1\} \subset \mathbb{F}_p$. How can one compute $u \wedge w$? xJsnark studies this problem; its solution is to split u and w into bits, do a bit-wise product, and recombine the bits of the result. The splits require 32 constraints each and the bit-wise product requires another 32 constraints, for 96 constraints in total. This is *very* expensive: xJsnark's SHA-256 implementation requires more than 20k constraints per block.

	Our design (multiple forms)			Baseline costs		
Operation	In form	Out form	Cost	Sparse	Bits	Instances
n-ary $+$	limbs, dense	limbs, dual	7	$2n+7$	≈32	176
aligned $\ggg$ / $\gg$	limbs, sparse	32bit, sparse	0	3	0	128
unaligned $\ggg$ / $\gg$	limbs, sparse	32bit, sparse	2	5	0	544
3-ary $\oplus$	32bit, sparse	limbs, dense	9	6	64	224
maj	limbs, sparse	limbs, dense	9	6	64	64
ch	limbs, sparse	limbs, dense	18	12	64	64
total			6064	8752	28160	

Table 1: SHA primitive operation costs in different forms. For our design, we give the input and output forms we use for each operation and the cost of applying that operation to a 32-bit word. (The cost of a shift or rotate is 0 if the offset aligns with a limb boundary, and 2 otherwise.) We also give the cost when values are all in limb-wise sparse form or represented as bits (as in xjSnark [73]), and the number of instances of each operation in the compression function. At the bottom, we give the net cost of all operation instances, which is a good approximation for SHA-256's total cost, per-round.

4.1 Word representations

Our efficient SHA-256 implementation begins with Bootle et al.'s *sparse form* [25]. Sparse form is an alternative to the natural *dense* form, which we have seen already. Let b_i be the bits of w, that is: $w = \sum_{i=0}^{31} 2^i b_i$. Then, w's dense form is $\mathsf{dense}_{32}(w) = \sum_{i=0}^{31} 2^i b_i \in \mathbb{F}$. Here, dense_{32} is a function from $\mathsf{u32} \to \mathbb{F}$. The sparse form of w is $\mathsf{sparse}_{32}(w) = \sum_{i=0}^{31} 4^i b_i \in \mathbb{F}$. The sparse form's bits are exactly the bits of w, with extra 0 bits interleaved.

Sparse form is useful because it makes checking many bitwise operations more efficient. For instance, one can show that for any two words u, w, the following holds, mod p:

$$\mathsf{sparse}_{32}(u) + \mathsf{sparse}_{32}(w) = 2\mathsf{sparse}_{32}(u \wedge w) + \mathsf{sparse}_{32}(u \oplus w)$$

Moreover, $u \wedge w$ and $u \oplus w$ are the only words that satisfy this equation. Thus, this fact gives a natural way to check XOR and AND relations, and similar equations can be used to check other bit-wise relations too.

One challenge when dealing with sparse and dense form is converting between them. This can be done through lookups to a table

$$\mathsf{DS}_b = \{(\mathsf{dense}_b(w), \mathsf{sparse}_b(w)) : w \text{ of } b \text{ bits}\}$$

where b is the bitwidth. For example, to convert a sparse representation x to dense form, P provides the dense equivalent y, and then shows that $(x, y) \in \mathsf{DS}_b$ using a lookup argument (S2). But, this approach is only efficient for small b, since lookup arguments for an n-size set of k-tuples have a one-time table setup cost n, which is 2^b here. So, we usually represent a word w *limb-wise*. For example, a 32-bit word is a 10-bit limb and two 11-bit limbs. This allows conversions to be done with tables of size 2^{10} and 2^{11}.

4.2 Our design

Our design is defined by the form we choose for each intermediate value in SHA. The compression function's inner loop manipulates registers $a, \ldots, h$. It performs bit-wise (rotate, xor, choice, majority) and arithmetic

(add) operations. Between iterations, the outputs (the adds) are written to two of the registers, and the registers are rearranged.

One of our insights is that the registers should actually be materialized in *both* sparse and dense form; that is in *dual* form. Re-arrangement causes registers to flow into an arithmetic operation (add) add in one iteration and into a bitwise operation (maj) in another, so having both forms makes both operation types cheap. Materializing both forms can cost a little, but as we will see, the tradeoff is worth it.

Table 1 shows the input and output forms we choose for each operation—and the induced cost. It also shows what the costs would be if we used only sparse form, or if we used the bit-wise encoding of xJsnark [73]. Our mixed-form approach beats a purely sparse approach because the savings we get from cheaper additions outweigh the marginal extra cost of materializing more values in dense form.

To optimize, we represent some values in 32-bit sparse form, instead of limb-wise. We do this for shift and rotation outputs. Since these operations move bits across limb boundaries, re-computing the limbs would require extra constraints. The "Sparse" column in Table 1 shows this cost. However, this limb-decomposition is actually unnecessary, because the only dependent operation is XOR. We find that XOR can be done efficiently encoded for 32-bit sparse inputs and a limb-wise dense output (Example 1). Thus, in our design, we omit this limb-decomposition for shift/rotation outputs. This makes shifts and rotations free if the offset is aligned with a limb boundary, and very cheap otherwise.

Example 1 (XOR constraints)**.** *Let u, v, w be 32-bit words. Let $u', v', w' \in \mathbb{F}_p$ be their 32-bit sparse encodings. The sum $s = u' + v' + w'$ does not overflow (assuming $p \geq 2^{64}$) and thus its even bits are exactly $u \oplus v \oplus w$.* P *provides field variables $(x_{0,\mathsf{o}}, x_{0,\mathsf{e}}), (x_{1,\mathsf{o}}, x_{1,\mathsf{e}}), (x_{2,\mathsf{o}}, x_{2,\mathsf{e}})$ which should be the odd and even bits of s, split into two 11-bit and one 10-bit limbs.* P *also provides the dense forms d_i of the $x_{i,\mathsf{e}}$.* P *shows the following lookups: $(d_0, x_{0,\mathsf{e}}), (d_1, x_{1,\mathsf{e}}) \in \mathsf{DS}_{11} \wedge (d_2, x_{2,\mathsf{e}}) \in \mathsf{DS}_{10} \wedge x_{0,\mathsf{o}}, x_{1,\mathsf{o}} \in \mathsf{S}_{11} \wedge x_{2,\mathsf{o}} \in \mathsf{S}_{10}$ (where $\mathsf{S}_b = \{\mathsf{sparse}_b(w) : w$ of b bits$\}$).* V *asserts the following constraint in $\mathbb{F}_p$: $u' + v' + w' = (x_{0,\mathsf{e}} + 2x_{0,\mathsf{o}}) + (2^{11})^2(x_{1,\mathsf{e}} + 2x_{1,\mathsf{o}}) + (2^{22})^2(x_{2,\mathsf{e}} + 2x_{2,\mathsf{o}})$.*

Since s contains the XOR in the even bits, one can show that the constraint and lookups ensure that (d_0, d_1, d_2) are the limbs of the XOR in dense form. The cost of this is 9 constraints (3 for 3 lookups into D_b, 6 for 3 lookups into DS_b, and one free linear constraint). Obtaining the XOR output in dense form is important because in SHA it is an input to a subsequent add.

Our SHA-256 design is similar to the halo2 library's SHA-256 implementation [60]. We discuss the similarities and differences in Section 9.

Extending to SHA-512. Our design naturally extends to SHA-512, which is used in EdDSA signatures (S7). To represent SHA-512's 64-bit words, we use 8 limbs per word. The widths of these limbs were chosen to maximize the number of bitwise shifts and rotations that can be performed for free (by rotating the limbs themselves). We also encode a slightly longer compression loop, because SHA-512 uses 80 rounds instead of SHA-256's 64.

5 RSA Signatures

In this work we focus on the most widely-used signature algorithm based on RSA, namely RSA-PKCS1v1.5. We depict the verification algorithm in Figure 2a. Verification consists of first computing the mod-N exponentiation D of the signature σ to the eth power, where (N, e) is the public key. Then, the SHA-256 hash of the message is encoded (along with DI, the DER-encoded algorithm identifier for the hash function [68]) using PKCS encoding as an integer D' mod N, and the result of the verification is the bit $D = D'$.

RSA signature verification has two steps which are difficult to represent in constraints—SHA-256 and the modular exponentiation. We can greatly reduce the cost of SHA-256 with Section 4's techniques. In this section we will discuss our new techniques for efficient representations of modular exponentiation, and

RSA.Vf(pk, σ, m):

$N, e \leftarrow \mathsf{pk}$

$t \leftarrow |N|$

$D \leftarrow \sigma^e \mod N$

$D' \leftarrow$

  PKCSEncode$(t, H(m))$

Return $D = D'$

PKCSEncode(t, h):

$\ell \leftarrow 3 + |h| + |\mathsf{DI}|$

$\mathsf{pad} \leftarrow \mathtt{FF} \cdots_{t-\ell} \mathtt{FF}$

$D \leftarrow$

  $\mathtt{0001}||\mathsf{pad}||\mathtt{00}||\mathsf{DI}||h$

Return D

(a) RSA-PKCS1v1.5 verification and encoding function. The value DI is a fixed string dependent on H. We assume $|N| > \ell$.

ECDSA.Vf(pk, σ, m):

$r, s \leftarrow \sigma$

If $r = 0$ or $s = 0$:

  Ret. false

$a \leftarrow \mathsf{H}(m)/s$; $b \leftarrow r/s$

$\hat{R} \leftarrow aG + b\mathsf{pk}$

Ret. $x(\hat{R}) = r \mod q$

ECDSA.Vf(pk, σ, m):

$R, z \leftarrow \sigma$

$r \leftarrow x(R) \mod q$

If $r = 0$ or $z = 0$:

  Ret. false

$h \leftarrow \mathsf{H}(m)$

Ret. $\mathsf{pk} = (-h/r)G + zR$

(b) Two equivalent versions of the ECDSA verification algorithm. The function $x(\cdot)$ extracts the x-coordinate from a point (x, y) on the curve.

Figure 2: RSA-PKCS1v1.5 and ECDSA verification algorithms.

modular arithmetic more broadly, in constraints. First, we explain how prior work reduces modular arithmetic to range checking (S5.1). Second, we explain how we reduce the costs of range checking (S5.2). Finally, we discuss other RSA-based signatures (S5.3). While our exposition is focused on RSA, our modular arithmetic optimizations in this section to all three signature schemes of interest.

5.1 From modular arithmetic to range checks

Using the standard square-and-multiply algorithm for modular exponentiation, we can reduce the task of modular exponentiation to modular multiplications and squarings. (In the case of RSA, because the exponent e is usually fixed to $65537 = 2^{16} + 1$, we even know the exact sequence of multiplications and squarings we must perform.) In constraint form, we can express modular multiplication as the constraint $ab = c \mod N$ on three values a, b, c. The key difficulty here is that this constraint must be checked only with additions and multiplications mod a prime p with $p < N$, where p is roughly 256 bits. Since N is 2048 bits, the values a, b, c must be expressed as $\ell = \lceil |N|/b \rceil$ "limbs" of b bits apiece, with multiplication done limb-wise.

Figure 3 show the architecture of a state-of-the-art limb multiplier [73] for two-limb inputs with 16 bits per limb. The high-level idea is to combine two modules for *polynomial multiplication* and *carry handling*. The overall inputs are $\vec{a} = (a_0, a_1)$ and $\vec{b} = (b_0, b_1)$. The overall output is $\vec{c} = (c_0, c_1, c_2, c_3)$. The output of the first module, $\vec{c'} = (c'_0, c'_1, c'_2)$, is the input to the second module.

The first module checks polynomial multiplication. For a limb vector $\vec{e}$ of length n, define the polynomial $f_{\vec{e}}(X) = \sum_{i=0}^{n-1} e_i X^i$. To satisfy the first module, the c'_i must be given values such that the following polynomial identity holds:

$$f_{\vec{a}}(X) \times f_{\vec{b}}(X) = f_{\vec{c'}}(X)$$

The module checks this identity at all $i \in \{0, \ldots, 2\}$. (Recall: two quadratic polynomials that agree at 3 points agree everywhere.)

Then, the second module handles carries. It ensures that when a carry-in o_i is added to the polynomial coefficient c'_i, the result has low-bits c_i and the high bits are carried out as o_{i+1}. The initial carry-in (o_0) is 0 and the final carry-out (o_3) is the result limb c_3. Both the carries and the c_i are asserted to be in small ranges; these assertions are called *range checks*. The carry ranges are slightly larger to deal with the potential size of limb overflow.

Somewhat surprisingly, the bottleneck when this system is translated into R1CS is carry-handling, not multiplication. Recall that linear operations are free, so the only costs are the 3 non-linear multiplications in the polynomial component and the range checks in the carry handler. These operations are shown in red in Figure 3.

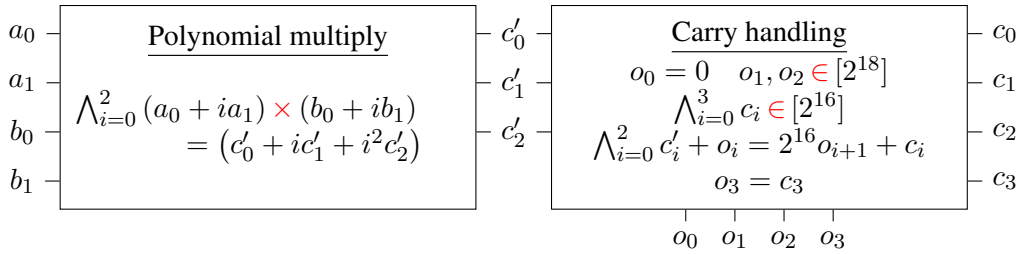

Figure 3: The (high-level) constraint system for testing limb-wise multiplication. It is described in detail in the main text. The expensive operations are in red and the range checks (set membership) are the bottleneck.

Of the two operation types, the range-checks are far more expensive in practice. We discuss the cost of range-checking in detail in the next section. But first, we give an example from prior work: in one state-of-the-art implementation [84] of RSA arithmetic, range checking is responsible for 94.5% of proving cost. So, improving range checking in I-R1CS is essential to achieve better performance.

5.2 Range checks through optimized lookups

Bit-splitting: the baseline. Prior RSA signature implementations range check with bit-splitting. In this approach, to range check $x \in \{0,\ldots,2^b-1\}$, one adds witnesses $x'_0,\ldots,x'_{b-1}$, and constraints $\sum_i x'_i 2^i = x \wedge \bigwedge_i x_i(x_i-1)=0$. The first of these constraints is linear (free), so the cost is b. In the concrete case of RSA signature exponentiation (Example 2), bit-splitting costs 83555 constraints.

Example 2. *In computing b^{65537} (mod a 2048-bit N), the implementation of Ozdemir et al. [84] uses 32-bit limbs. It performs 2176 32-bit range checks (for initial limbs) and 357 39-bit range checks (for carries).*

Lookups: a promising theory. One tool for (theoretically) reducing the costs of range checks is a *lookup proof*. A lookup proof shows that some $x \in \mathbb{F}$ is in a set $V \subseteq \mathbb{F}$ (V is called a *table*). State-of-the-art lookup proofs [62], incur a one-time table setup cost of $|V|$ constraints; thereafter, each membership claim (each *lookup*) costs only one constraint.

At first glance, lookup proofs may seem to easily resolve the bottleneck in modular arithmetic. Simply set $V=\{0,\ldots,2^b-1\}$ and replace each cost-b bit-split with a cost-1 lookup. This approach makes each range check cheap, but the table setup costs are astronomical. For example, setting up the table for the 32-bit range checks in Example 2 would incur a cost of 2^{32} constraints. Using lookups effectively requires slightly more ingenuity.

Optimizing lookups for range checking. The first issue with lookups, is that the table setup cost is exponential in the number of bits. We reduce this cost using the natural idea of splitting a large range into sub-ranges: to range check an element x fits into b bits, express x in base $2^{b'}$ for some b' that divides b and check that each of the smaller "sub-limbs" $x[i]$ has b' bits. We then combine the smaller parts (at no constraint cost) to retrieve x. With subranges, the setup costs $2^{b'}$ constraints and each b-bit range check costs b/b' constraints. (If $b' \nmid b$, we need only one additional constraint to range check the remaining $b \mod b'$ bits.)

A second, confounding, issue is the need to range-check multiple different ranges in a given circuit. For instance, in an RSA implementation (Example 2), we must range check the limbs of the inputs (32 bits) as well as intermediate carries that arise from limb-wise operations (39 bits). Considering each limb size separately is sub-optimal: it misses potential opportunities to *share* subrange setup costs between the different sizes. Thus, choosing optimal subrange sizes is a global optimization problem, which we state in Definition 3.

A third issue is the fact some range-checks are *flexible*. We first explain this in the context of Example 2. Since its carries can be as large as 39 bits, forcing them to fit in a smaller range will make the constraint

system incomplete (unsatisfiable for some legitimate inputs). However, the soundness of the constraint system does not strictly require the carries to fit in 39 bits. The only requirement is that certain equations involving the carries should not overflow; which is guaranteed even if the carries fit in 50 bits.

Considering all three issues—setup costs, shared tables, and flexible bitwidths—we pose an optimization problem:

Definition 3. *Let l_i be a bit lower bound, u_i be a bit upper bound, and n_i be the number of values claimed to be in $\{0, \ldots, 2^{b_i} - 1\}$, with $b_i \in \{l_i, \ldots, u_i\}$. A **batched flexible range check problem** P is a sequence $(l_1, u_1, n_1), \ldots, (l_k, u_k, n_k)$. A **subrange sizing scheme** for P is a is a sequence of multisets $S = S_1, \ldots, S_k$ such that each S_i sums to a value in $\{l_i, \ldots, u_i\}$. The (constraint) cost of S is $\sum_{s \in \cup_i \mathsf{set}(S_i)}(2^s) + \sum_i |S_i| n_i$.*

To solve this problem, we design a planner to find good sizing schemes for a problem P. For our problems of interest, a simple brute-force grid search easily finds the plan that minimizes total constraint cost.

For Example 2, expressed as $(l_1 = 32, u_1 = 32, n_1 = 2176)$ and $(l_2 = 39, u_2 = 50, n_2 = 357)$, our planner returns $S_1 = \{\!\{8, 8, 8, 8\}\!\}, S_2 = \{\!\{8, 8, 8, 8, 8\}\!\}$, which has cost 10745. This is almost an $8\times$ improvement over bit-splitting. Notice that in this solution, the two kinds of ranges are sharing a single subrange of 8 bits.

We envision that our approach of globally optimizing subrange sizes for range checks will be useful in other uses of modular arithmetic within proof systems. Our planner does not currently consider optimizations *across* the hash and modular multiplication subcircuits; doing this is an interesting direction we leave to future work.

5.3 RSA-PSS signatures

Some applications use RSA with the PSS padding scheme instead of PKCS1v1.5 [15, 65]. PSS is more complex but yields a scheme with a tight reduction to the hardness of inverting RSA. Our approach can be generalized to work with PSS as well as PKCS1v1.5, but doing PSS padding verification in R1CS would incur some additional costs, mostly stemming from the hashes done in the "mask generation function" (MGF). Concretely, we estimate that PSS padding would cost roughly 63,000 additional R1CS constraints for a 2048-bit modulus and using SHA-256 for the MGF. We leave implementing and optimizing this approach to future work.

6 ECDSA Signatures

The Elliptic Curve Digital Signature Algorithm, or ECDSA, is a signature scheme over an elliptic curve group $\mathbb{G}$ of order q with generator G over a finite field $\mathbb{F}_p$ for a prime p. The most common choice of $\mathbb{G}$ in applications is the NIST standard curve secp256r1 (also called P-256) [74]. We focus on this curve in this work, but our techniques are fully general to other curve choices. We depict two mostly equivalent verification algorithms in Figure 2b (with slightly different inputs). Our discussion below will focus on the right-hand equation for concreteness, but it applies to both. To prove possession of an ECDSA signature (R, z) on message m, the prover needs to prove the following relation:

$$\mathcal{R}_{ECDSA} = \left\{ \begin{array}{l} (G, \mathsf{pk}; (r, R, z, m)) : \\ \mathsf{pk} = (-H(m)/r)G + zR \wedge \quad r = x(R) \mod q \quad \wedge r \neq 0 \quad \wedge z \neq 0 \end{array} \right\}.$$

As with RSA-PKCS1v1.5 verification (S5) hashing is a bottleneck in representing ECDSA verification in R1CS; we can apply the techniques of S4 to improve performance. The other major bottleneck is the three elliptic curve operations that comprise the verification equation $\mathsf{pk} = (-h/r)G + zR$—two scalar

multiplications to compute $(-h/r)G$ and zR, and a point addition to combine them. (There are techniques, such as Straus [102], to generically reduce the cost of computing the right-hand side of the equation. Looking ahead, our sidecar protocol, described in S6.1, will obviate the potential benefits of Straus, so we will not discuss it further below.)

Scalar multiplications are, as with modular exponentiations, implemented via a double-and-add chain that iterates over the bits of the scalar and performs a point double (if the bit is 0) and also a point addition (if it is 1). P-256 scalars are 256 bits, and since we must pay for both branches in R1CS, a single scalar multiplication requires 256 additions and 256 doublings. A single point addition requires several operations in the base field $\mathbb{F}_p$, including four costly reductions mod p. If doublings are implemented via addition, each iteration of the double-and-add chain requires eight modular reductions.

A useful comparison is to the modular exponentiation circuit for RSA: this requires only one reduction mod N per bit of the public exponent e, which is fixed to have only seventeen bits. Thus, only two iterations of double-and-add have nearly as many modular reductions (though with a smaller modulus) as the entire RSA modular exponentiation circuit. This points to the drastic difference in cost in the prior state of the art for RSA vs. elliptic curves: for xJsnark, modular exponentiation only cost about 89k R1CS constraints, vs. 687k constraints for a single scalar multiplication for P-256. (xJsnark did not report an ECDSA verification benchmark, but we estimate the cost would be at least 1.4 million constraints excluding hashing.)

Because of this, a natural first step for improving the efficiency of ECDSA is applying the lookup-table-based improvements to modular arithmetic discussed in S5 to elliptic curve operations. This greatly improves our performance—a naive baseline implementation of scalar multiplication costs 1.3 million constraints; the same implementation with lookup-based range checks only costs 277k constraints.

Preprocessing and incomplete formulas. We can make more progress by exploiting the fact that scalar multiplications for a fixed base point can be accelerated using preprocessing. The P-256 generator G is fixed, so the computation of $(-h/r)G$ can use preprocessing. We use a variant of the classic "window method" [1, 71] which works as follows. For scalar bit length n, pick a chunk bit length w and let $\ell = \lceil n/w \rceil$. During preprocessing, compute the table G_{pow} such that $G_{\text{pow}}[i][j] = j \cdot 2^{wi} \cdot G$ for $i \in [0, 1, \cdots, \ell]$ and $j \in [1, \cdots, 2^w)$. Then, during scalar multiplication, write the scalar x as a sequence of base-2^w chunks $(\text{chunk}_i)_{i \in [0,\ell-1]}$, so that $x = \sum_{i \in [0,\ell-1]} \text{chunk}_i * 2^{w*i}$. The output of the scalar multiplication is $\sum_{i \in [0,\ell-1]} G_{\text{pow}}[i][\text{chunk}_i]$. This method's cost is ℓ additions and lookups into a size $\ell \cdot 2^w$ table.

We further optimize this approach using the observation that because the P-256 group has prime order, the edge cases that must be handled in the "complete" formula for adding two points P_1 and P_2—e.g., $P_1 = \pm P_2$—cannot arise in scalar multiplication, with or without the window method, except possibly in the last iteration. This is roughly because in each iteration except the last one, two distinct non-identity points of order much less than q are added together; because no wraparounds mod q occur, the result (and input to the next iteration) is also a non-identity point of order much less than q. This observation has been made in other literature on optimized elliptic curve implementations [2, 26], but as far as we know we are the first to use it to optimize R1CS representations of elliptic curve operations. (Looking ahead to S6, for Ed25519 verification, complete vs. incomplete formulas are a nonissue, since by design the curve has a complete and branchless point addition formula [19].) For fixed base points, these improvements bring the constraint cost down to 65k constraints (from 277k).

6.1 A "Sidecar" Protocol for ECDSA

Our optimizations for elliptic curve operations greatly reduce the constraint cost of R1CS verification. However, the constraint cost is still quite large—around 350k constraints for full ECDSA verification, excluding hashing. The main bottleneck is still elliptic curve operations, and in particular the non-fixed-base scalar multiplication zR (which costs ≈277k). In a proof of possession, the base R even must be hidden

$\underline{P^H(\mathsf{pp}; \mathsf{pk}; (R, z, m))}$:

$((G, K), \mathsf{pp}_{zk}) \leftarrow \mathsf{pp}$; $v, f_1, f_2, f_3 \leftarrow\!\!\$\ \mathbb{F}_q$

$C^{(1)} \leftarrow vK + R$; $U \leftarrow f_1 C^{(1)} - f_2 K - f_3 G$

$r \leftarrow x(R) \mod q$; $o, o' \leftarrow\!\!\$\ \mathcal{O}$

$e_1 \leftarrow z$; $e_2 \leftarrow e_1 v$; $e_3 \leftarrow H(m)/r$

$\mathsf{cm} \leftarrow \mathsf{HCom}\left((e_i)_{i\in[3]}, o\right)$

$\mathsf{cm}' \leftarrow \mathsf{HCom}\left((f_i)_{i\in[3]}, o'\right)$

$c \leftarrow H(\mathsf{pk}, C^{(1)}, U, \mathsf{cm}, \mathsf{cm}')$

$s_i \leftarrow f_i + e_i c \quad \forall i \in [3]$

$\mathbb{x} \leftarrow \left(C^{(1)}, \mathsf{cm}, \mathsf{cm}', c, (s_i)_{i\in[3]}\right)$

$w \leftarrow \left((e_i, f_i)_{i\in[3]}, o, o', v, m, R\right)$

$\pi_{zk} \leftarrow \mathsf{Prv}(\mathsf{pp}_{zk}; (G, K); \mathbb{x}; w)$

Return $\pi = (U, \mathbb{x}, \pi_{zk})$

(a) The proof generation algorithm for $\Pi^{FS}_{\mathsf{ECDSA}}$.

$\underline{V^H(\mathsf{pp}; \mathsf{pk}, \pi)}$:

$((G, K), \mathsf{pp}_{zk}) \leftarrow \mathsf{pp}$

$(U, \mathbb{x}, \pi_{zk}) \leftarrow \pi$

$(C^{(1)}, \mathsf{cm}, \mathsf{cm}', c, (s_i)_{i\in[3]}) \leftarrow \mathbb{x}$

$c \leftarrow H(\mathsf{pk}, C^{(1)}, U, \mathsf{cm}, \mathsf{cm}')$

Return $s_1 C^{(1)} - s_2 K - s_3 G \stackrel{?}{=} U + c \cdot \mathsf{pk}$

$\quad \wedge \quad \mathsf{PVer}\left(\mathsf{pp}_{zk}; \mathbb{x}; \pi_{zk}\right) \stackrel{?}{=} 1$

(b) The proof verification algorithm for $\Pi^{FS}_{\mathsf{ECDSA}}$.

Figure 4: The proof generation and verification algorithms for our sidecar protocol $\Pi^{FS}_{\mathsf{ECDSA}}$ correspond to the relation $\mathcal{R}_{ECDSA}$, and the inner SNARK uses the relation $\mathcal{R}_{\mathsf{ARK}}$ (both relations defined in the main text). The function H is the hash used for Fiat-Shamir.

from the verifier. Using the other ECDSA verification still requires computing $b\mathsf{pk}$; while pk is public it may not be fixed, making it difficult to use preprocessing.

Thus, to reduce costs further, we introduce a protocol that completely eliminates the cost of the non-fixed-base scalar multiplication. This protocol works by moving the verification check $\mathsf{pk} = (-h/r)G + zR$ outside R1CS completely, and into a separate sigma protocol that is run alongside the "main" ZKP protocol. We depict our protocol in Figure 5, and explain it next.

Warmup: Okamoto's protocol and Sigmabus. To understand how our protocol works, it is useful to first think of a relaxed setting in which we can reveal the point R to the verifier. To prove possession of a signature on message m, it would suffice to prove knowledge of a pair of scalars (x, y) so that $x \neq 0$, $y = H(m)$, and (x, y) is a representation of pk with respect to points R and $U = (-1/r)G$. In other words, it would suffice to prove the relation

$$\mathcal{R}_{public} = \left\{ \begin{array}{l} (\mathsf{pk}, R, U; (m, x, y)) : \\ \mathsf{pk} = xR + yU \\ \wedge \quad x \neq 0 \wedge H(m) = y \end{array} \right\}$$

The first constraint has an efficient sigma protocol, due to Okamoto [81]; standard techniques can also prove the second constraint. A recent work called Sigmabus [67] showed a surprisingly simple approach to "bridge" sigma protocol executions and SNARKs. Roughly, their approach works by modifying the sigma protocol so the prover's commitment (i.e., its first message) includes a ZKP-friendly commitment to the witnesses, then requiring the prover to prove (1) these witnesses were the same as witnesses given to some other ZKP prover, and (2) that its sigma protocol response was computed correctly using these witnesses. This approach lets the verifier check statements about these witnesses with a sigma protocol, a SNARK, or both. Applying this to our relation would allow the prover to prove the first two constraints using cheap sigma protocols, the third constraint $H(m) = y$ with a SNARK, and show that the y used in both steps was the same, thus linking them together. This would not be zero-knowledge since R is revealed.

Hiding R. The main idea for fixing this is as follows. We can use most of the ideas from the previous protocol, but just replace R with a commitment to R. To preserve the ability to verify the elliptic curve equation with a sigma protocol, though, the commitment must respect the structure of the curve group. Rather than standard schemes like Pedersen commitments, which only commit to $\mathbb{F}_p$ elements, we use a version of ElGamal encryption: for a fixed generator K of $\mathbb{G}$, and hash function HCom, our scheme is $\mathsf{Com}_K(R, (v, o)) := (\mathsf{HCom}(v, o), vK + R)$. The "public key" K need only have an unknown discrete log to the base G, so it can easily be generated via hash-to-curve. Recall our original verification equation $\mathsf{pk} = (-h/r)G + zR$. If we let $(C^{(0)}, C^{(1)}) = \mathsf{Com}_K(R, (v, o)) = (\mathsf{HCom}(v, o), vK + R)$, then we can rewrite $R = C^{(1)} - vK$ and plug it into our verification equation to get $\mathsf{pk} = (-h/r)G + z(C^{(1)} - vK) = zC^{(1)} - zvK - (h/r)G$. The right-hand side of this is a representation of pk with scalars $(e_1, e_2, e_3) = (z, zv, h/r)$ and three *public* group elements $C^{(1)}, -K, -G$ as bases. We thus simply prove knowledge of this representation using a three-base version of the same Okamoto protocol as described above. We use Sigmabus's transform to provide the representation as a witness to the SNARK; this is a crucial step to allow verifying that $e_2 = e_1 v$ for the v committed in $C^{(0)}$ and $e_3 = h/r$. Finally, to ensure the prover's provided value of r is indeed equal to $x(R) \mod q$, we must compute $C^{(1)} - vK$ in the circuit. Thus, we must still do one scalar multiplication in constraints; however, the base K is fixed and public, so preprocessing can reduce the cost.

Summary. Our protocol for proving possession of an ECDSA-P256 signature (R, z) on message m for public key pk is in Figure 4. It leverages a sigma sub-protocol and a SNARK sub-protocol. The sigma sub-protocol proves that pk represents in a specific form relative to $C^{(1)}$, $-K$ and $-G$. The SNARK sub-protocol proves the remaining constraints, specifically for the relation $\mathcal{R}_{\mathsf{ARK}}$ defined as the set of tuples $\left(K; \left(C^{(1)}, \mathsf{cm}, \mathsf{cm}', c, (s_i)_{i\in[3]}\right); \left((e_i, f_i)_{i\in[3]}, o, o', m, R\right)\right)$ satisfying the following constraints:

$$
\begin{aligned}
& s_i = f_i + e_i c \;\forall\, i \in \{0,1,2\} \\
\wedge \quad & \mathsf{cm}' = \mathsf{HCom}\left((f_i)_{i\in[3]}, o'\right) \wedge \mathsf{cm} = \mathsf{HCom}\left((e_i)_{i\in[3]}, o\right) \\
\wedge \quad & C^{(1)} = vK + R \quad \text{where } v = e_2 e_1^{-1} \mod q \\
\wedge \quad & e_3 = H(m)/r \quad \wedge \quad e_1 \neq 0 \\
\wedge \quad & r \neq 0 \quad \text{where } r = x(R) \mod q
\end{aligned}
$$

The protocol in Figure 4 contains some optimizations: first, we commit the representation (e_1, e_2, e_3) and the random scalars (e_1', e_2', e_3') in one hash apiece, instead of committing to each individually. We also use the commitments of e_1 and e_2 as a commitment to v instead of $C^{(0)}$; these two commitments already bind v. We describe the prover in prose here. To verify the proof $U, (s_1, s_2, s_3), \pi_{zk}$ for statement $\mathbb{x}$ and pk, the verifier first checks $s_1 C^{(1)} - s_2 K - S_3 G \stackrel{?}{=} U + c\mathsf{pk}$, then checks $\mathsf{PVer}\left(\mathsf{pp}_{zk}; \mathbb{x}; \pi_{zk}\right)$ and returns the logical AND of the two results. In Appendix A, we analyze the security of this protocol.

Theorem 2. *The protocol $\Pi^{FS}_{\mathsf{ECDSA}}$ for proving possession of an ECDSA-P256 signature is complete, adaptively knowledge sound and non-interactive zero-knowledge if the following conditions hold:*

1. HCom *is a binding and hiding commitment scheme, and*
2. SNARK *is non-interactive zero-knowledge, with its interactive version being special sound.*

6.2 ECDSA Ring Signatures

Though it is not our main focus in terms of functionality, we observe that the sidecar protocol from S6.1 leads to an efficient protocol for proving possession of an ECDSA-P256 signature under a committed pk. This functionality is interesting in particular because it can be used to build ring signatures. (We give more detail on this below.)

$\underline{P^H((L,K);(E_0,E_1,R);(z,z',t,o)):}$

$f_1, f_2, f_3, f_4 \leftarrow\!\!\$\ \mathbb{Z}_q$
$U_1 \leftarrow f_1 L$
$U_2 \leftarrow f_2 R + f_1 K$
$U_3 \leftarrow f_3 E_0 + f_4 L$
$U_4 \leftarrow f_3 E_1 + f_4 K$
$c \leftarrow H((E_0, E_1, R), U_1, U_2, U_3, U_4)$
$s_1 \leftarrow f_1 + vc$
$s_2 \leftarrow f_2 + zc$
$s_3 \leftarrow f_3 + z'c$
$s_4 \leftarrow f_4 - tc$
Return $\pi = ((U_1, U_2, U_3, U_4), (s_1, s_2, s_3, s_4))$

(a) The proof generation algorithm.

$\underline{V^H((L,K);(E_0,E_1,R,\pi)):}$

$((U_1, U_2, U_3, U_4), (s_1, s_2, s_3, s_4)) \leftarrow \pi$
$c \leftarrow H((E_0, E_1, R), U_1, U_2, U_3, U_4)$
Return $s_1 \cdot L \stackrel{?}{=} U_1 + (c \cdot E_0)$
$\quad \wedge \quad s_2 \cdot R + s_1 \cdot K \stackrel{?}{=} U_2 + (c \cdot E_1)$
$\quad \wedge \quad s_3 \cdot E_0 + s_4 \cdot L \stackrel{?}{=} U_3$
$\quad \wedge \quad s_3 \cdot E_1 + s_4 \cdot K \stackrel{?}{=} U_4 + c \cdot R$

(b) The proof verification algorithm.

Figure 5: The proof generation and verification algorithms for the non-interactive variant of our sigma protocol Π_{rs} for relation $\mathcal{R}'_{rs}$ defined in Section 6.2. The function H is the hash used for Fiat-Shamir.

Following prior work [37, 54] on ring signatures for ECDSA, the exact relation we will target is one where the public key is committed, the message and presignature R are known, and the scalar z is hidden. We formally define the relation $\mathcal{R}_{rs}$ as a set of tuples $((G, \mathsf{pp}), (\mathsf{cm}_{\mathsf{pk}}, m, R), (z, \mathsf{pk}, o))$ satisfying the following constraints: (1) $\mathsf{cm}_{\mathsf{pk}} = \mathsf{Com}_{\mathsf{pp}}(\mathsf{pk}, o)$, (2) $\mathsf{pk} = zR - (h/r)G$ where $h = H(m)$ and $r = x(R) \mod q$, (3) $z \neq 0$.

Our main insight, which is also used in the sidecar protocol above, is that by using ElGamal encryption with a fixed public key as the commitment scheme we can preserve the algebraic relationship between pk, R, G, allowing the use of cheap sigma protocols to prove that the verification equation holds. We define the commitment scheme to pk with randomness o as $\mathsf{Com}'(\mathsf{pk}, o) = (C^{(0)}_{\mathsf{pk}}, C^{(1)}_{\mathsf{pk}}) = (oL, oK + \mathsf{pk})$ for $L, K \in \mathbb{G}$. (As above, we can generate L, K transparently using hash-to-curve.)

With this commitment scheme, and adding L, K to the public parameters, we can replace constraint (1) of $\mathcal{R}_{rs}$ with $C^{(0)}_{\mathsf{pk}} = oL$ and transform constraint (2) to $C^{(1)}_{\mathsf{pk}} + (h/r)G = zR + oK$ where $h = H(m)$. These two transformed equations are linear, and their base points $E_0 := C^{(0)}_{\mathsf{pk}}, E_1 := C^{(1)}_{\mathsf{pk}} + (h/r)G$, L, R, K are public. We can thus prove (1) and (2) using standard sigma protocols. For the last constraint $z \neq 0$, we express it as $z \cdot z' = 1$ for some $z' \in \mathbb{Z}_q$. This is because $z \in \mathbb{Z}_q$, and only non-zero elements in $\mathbb{Z}_q$ have multiplicative inverses. We transform the non-linear equation $z \cdot z' = 1$ into linear constraints by leveraging (1) and (2). Specifically, we raise both sides of those equations by z' to obtain the following two new equations in a new variable $t = oz'$: (1) $E_0^{z'} = tL$ and (2) $z'E_1 = R + tK$.

The final transformed relation $\mathcal{R}'_{rs}$ is as follows:

$$\mathcal{R}'_{rs} = \left\{ \begin{array}{l} ((L,K),(E_0,E_1,R),(z,z',t,o)) : \\ \quad E_0 = oL \quad \wedge \quad E_1 = zR + oK \quad \wedge \quad z'E_0 = tL \quad \wedge \quad z'E_1 = R + tK \end{array} \right\}.$$

We give a sigma protocol for this relation in Figure 5b. By leveraging our sigma protocol for $\mathcal{R}'_{rs}$, we can construct a proof of knowledge for the relation $\mathcal{R}_{rs}$: first, the prover and verifier locally compute $r \leftarrow x(R) \mod q$, $E_0 \leftarrow C^{(0)}_{\mathsf{pk}}$ and $E_1 \leftarrow C^{(1)}_{\mathsf{pk}} + (h/r)G$. The prover computes $z' \leftarrow z^{-1} \mod q$ and $t \leftarrow oz'$. Then, the prover and verifier engage in our sigma protocol with common inputs $((L,K),(E_0,E_1,R))$ and prover witnesses (z, z', t, o).

Building ring signatures. Our proof of knowledge Π_{rs} can be used to build an ECDSA-based ring signature with significantly better performance than with prior work [37, 54]. Suppose that we want to prove that the prover has a valid ECDSA signature on a public message m corresponding to one of the verification keys on a given list $\mathcal{S}$. A set of tuples $((G, \mathsf{pp}, \mathcal{S}), (\mathsf{cm}_{\mathsf{pk}}, m, R), (z, \mathsf{pk}, o))$ is in the relation $\mathcal{R}_{ring}$ if the following constraints are satisfied: (1) $\mathsf{cm}_{\mathsf{pk}} = \mathsf{Com}_{\mathsf{pp}}(\mathsf{pk}, o)$, (2) $\mathsf{pk} = zR - (h/r)G$, (3) $z \neq 0$, and (4) $\mathsf{pk} \in \mathcal{S}$. This relation can be split into two relations: $\mathcal{R}'_{rs}$ and $\mathcal{R}_{memb}$, where $\mathcal{R}'_{rs}$ is the same before and the relation $\mathcal{R}_{memb}$ is defined as the set of tuples $\big((G, \mathsf{pp}), (\mathsf{cm}_{\mathsf{pk}}, \mathcal{S} = (\mathsf{pk}_0, \cdots, \mathsf{pk}_{N-1})), (\ell, o)\big)$ so that: (1) $\ell \in \{0, \ldots, N-1\}$, (2) $\mathsf{cm}_{\mathsf{pk}} = \mathsf{Com}_{\mathsf{pp}}(\mathsf{pk}_\ell, o)$.

To generate a proof for the relation $\mathcal{R}'_{rs}$, the prover utilizes Π_{rs}. For the relation $\mathcal{R}_{memb}$, we employ the Groth-Kohlweiss protocol [61], designed to prove that one out of many commitments can be opened to 0, and that the prover knows such an opening. This protocol requires the underlying commitment scheme to be additively homomorphic, but our ElGamal-like commitment scheme Com' is not. To address this mismatch, we observe that for every message M in the message space $\mathcal{M}$ of Com', there must exist a scalar m such that $mG = M$ with G as a generator. This observation allows us to treat Com' as an additively homomorphic commitment scheme over the transformed message space

$$\mathcal{M}' = \{\, m \;:\; mG = M \quad \forall M \in \mathcal{M} \,\}.$$

We now present a protocol Π_{memb} for the relation $\mathcal{R}_{memb}$. Given a commitment $\mathsf{cm}_{\mathsf{pk}} = (C_0, C_1)$ on the public key pk, we define a list of commitments to be $\mathcal{S}' = (C_0, C_1 - \bar{pk})_{\bar{pk} \in \mathcal{S}}$. Using Groth-Kohlweiss protocol, the prover shows that he knows how to open one of these commitments to 0. The security of Π_{rs} and the Groth-Kohlweiss protocol, together with the binding property of Pedersen Commitment guarantees that, if the prover can generate valid proofs for relation $\mathcal{R}_{rs}$ and $\mathcal{R}_{memb}$ respectively with the same $\mathsf{cm}_{\mathsf{pk}}$, then the prover has a valid ECDSA signature on m corresponding to one of the verification keys on the list $\mathcal{S}$. We analyze the security of our construction in Appendix B.

7 EdDSA Signatures

The Edwards-curve Digital Signature algorithm (EdDSA) is an elliptic-curve-based signature scheme, standardized in RFC 8032. It uses a derandomized version of classic Schnorr signatures over an elliptic curve $\mathbb{G}$. The most common choice of $\mathbb{G}$ is Ed25519, an Edwards-type curve over the base field $\mathbb{F}_p$ for $p = 2^{255} - 19$ which is birationally equivalent to Curve25519 [18, 19]. The order of $\mathbb{G}$ is $8q$ for a large prime q. An Ed25519 signature is a pair $(R, s) \in \mathbb{G} \times \mathbb{F}_q$. To verify a signature on message m for public key pk, the verifier first computes $c \leftarrow H(R, \mathsf{pk}, m)$ then returns the result of the check $sG = R + c\mathsf{pk}$. (We omit input validity checks, encoding, and cofactors here for simplicity.)

All techniques from the previous sections—efficient hashing (S4) and modular arithmetic (S5), preprocessing (S6), and sidecar protocols (S6.1)—can be applied to improve the efficiency of EdDSA proofs of possession. (We omit the details for brevity, but it is straightforward to adapt our sidecar protocol to EdDSA, since EdDSA's group operations are similar to ECDSA's.)

Applying all these techniques gives a huge reduction in constraint cost. Without these techniques, our baseline implementation for EdDSA verification of a 64-byte message required approximately 3.3 million constraints (of which roughly 95% correspond to the algebraic portion) and about 18 seconds to produce a proof. With all our techniques, the number becomes only 73k constraints, a 45× improvement. However, we still have one relatively expensive operation left: namely, the fixed-based scalar multiplication needed to obtain the group element R in R1CS. Even with preprocessing this operation is the dominant cost in the EdDSA verification circuit. As we described above, the main bottleneck of fixed-base scalar multiplication is emulating non-native $\mathbb{F}_p$ (i.e., the curve's base field) operations in the native R1CS field $\mathbb{F}_q$. To remove this last bottleneck we employ a simple, but powerful, idea—changing the R1CS field to the "right" one, $\mathbb{F}_p$.

$p = 2^{256} - 45102495181026339137933092255703437487 7$
E: $y^2 = x(x^2 + ax + b)$, with:
$a =$ `0x83D55B3EF1207CBB74ADA704E61ADF4DABAED20EAE494CC45293FDCEFDD1183D`
$b =$ `0x341B58146036CB9911638F4CF4AC3BED671E867F1B14831C1AF9CD915591B64C`
Order of $E/\mathbb{F}_p$ is $2 * (2^{255} - 19)$

Figure 6: Parameters for our double-odd curve T-25519

7.1 "Right-field" Arithmetic

This idea is folklore and has been explored in other works. However, as we will see, applying right-field arithmetic to improve performance for ECDSA or Ed25519 verification is nontrivial; further, the technique has some critical limitations.

The main challenge is constructing a group (for efficiency reasons, this is always an elliptic curve group) where discrete log is hard but whose order (or a subgroup's order) is equal to p, the curve's base field modulus. This is needed to instantiate the polynomial commitment used by the most efficient ZKPs, and in particular Mirage and our extension of Spartan (S3). These protocols work by (roughly) transforming R1CS satisfiability into a statement about polynomials over $\mathbb{F}_p$, committing to these polynomials, then convincing the verifier some relationships between them hold.

Generating an elliptic curve group with a given order is complex, but has some known solutions. A generic method to make a curve of order p is to find a prime r such that $p = r + 1 - t$ and $t^2 - 4r = DV^2$ (equivalently, $(t-2)^2 - 4p = DV^2$) for some integers D and V with D being negative and equal to 0 or 1 modulo 4. If D's absolute value is small enough, then the Hilbert class polynomial of discriminant D can be computed in practice; it splits over $\mathbb{F}_r$, the roots being the j-invariants of curves of order exactly p over $\mathbb{F}_r$.

For P-256, prior work [54] used a version of this method to generate a curve, which they call T-256, whose group has order equal to P-256's base field modulus. We use their curve to implement right-field arithmetic for ECDSA over P-256. In the case of Ed25519, no equivalent curves were previously known. Prior work [4] used a different method to generate a *pairing-friendly* curve over a 574-bit field with a subgroup of order $p = 2^{255} - 19$; the pairing-friendliness is not needed for our use case and would only hurt our performance. Instead, we used the method above—specifically algorithm 3.2 from [97]—to generate a novel elliptic curve of order $2p$ over a 256-bit field. For the sake of symmetry, we call this curve T-25519. Its parameters are listed in Figure 6. T-25519 has discriminant $D = -2330728$ and is a double-odd curve, allowing the definition of a group of order exactly p, with fast and complete formulas and canonical encoding [91]. Our T-25519 curve is much faster than the pairing-friendly curve: on an Intel "Coffee Lake" 2.3GHz CPU, T-25519 was 5-5.7 times faster for scalar and multi-scalar multiplication and only about one-third slower than an optimized P-256 implementation. Right-field arithmetic has a dramatic effect on constraint costs: a baseline fixed-base scalar multiplication costs 632k constraints; with right-field arithmetic it costs only ≈3200 (a 200× reduction).

Limitations. Right-field arithmetic has a critical limitation. Because the base fields of P-256 and Ed25519 do not have points of multiplicative order a large power of two, proof systems that that assume this structure will not be efficient with right-field arithmetic. This includes Groth16 and Mirage, one of our backend proof systems of interest. These systems need this structure to use FFT-based polynomial algorithms (e.g. multiplication). Other more general smoothness conditions are also not met by the P-256 and Ed25519 base fields. (Theoretical fast polynomial algorithms, such as ECFFT, are unlikely to be concretely efficient enough for our purposes.) Thus, right-field prover performance with these backends is likely to be very poor. This means right-field arithmetic cannot be combined with backends like Groth16 that produce very small proofs. In settings where larger proofs are permissible, we envision a backend like Dorian (S3) being used. Also, in settings where compliance with government standards like FIPS is needed, using a customized elliptic curve

	Ours (Mirage)			**Baseline** (Groth16)		
$\|m\|$ (B)	64	512	2048	64	512	2048
|R1CS| (k)	13	58	209	29	231	925
P time (s)	0.2	0.7	2.6	0.2	1.0	3.8

(a) Comparison between our SHA-256 implementation and Z#'s standard library. All verifier times were less than 12 milliseconds. Proof sizes were 240B for ours and 192B for the baseline.

	Ours (Mirage)			**xJsnark** [73]	**Circom** [116]
$\|m\|$ (B)	64	512	2048	3	0
|R1CS| (k)	40	84	235	89	536
Ptime (s)	0.7	1.3	3.0	2.3	15.9

(b) Proof of RSA-PKCS1v1.5 signature possession, with 2048-bit keys. Our verifier times were 28 milliseconds. Our proof sizes were 240B, others were 128B.

	Ours (Mirage)				**Ours** (Dorian)				**halo2** [115]
$\|m\|$ (B)	0	64	512	2048	0	64	512	2048	0
|R1CS| (k)	25	39	83	234	8	21	65	216	N/A
P time (s)	0.3	0.5	1.0	2.7	0.2	0.4	0.9	2.5	2.2
V time (s)	.02	.03	.03	.03	0.2	0.2	0.2	0.4	.07
$\|\pi\|$ (kB)	0.5	0.5	0.5	0.5	19	31	32	51	6.5

(c) Proof of ECDSA-P256-SHA256 signature possession.

	Ours (Mirage)				**Ours** (Dorian)				**halo2** [11]
$\|m\|$ (B)	0	64	512	2048	0	64	512	2048	0
|R1CS| (k)	30	73	128	344	9	53	107	323	N/A
P time (s)	0.6	0.9	1.5	4.2	0.2	1.0	1.7	4.8	4.5
V time (s)	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.3	0.01
$\|\pi\|$ (kB)	0.6	0.6	0.6	0.6	15.4	29.7	46.8	81.0	5.6

(d) Proof of Ed25519 signature possession.

Figure 7: Experimental results of our proof of possession implementations. Results are on our server testbed. We show message length, constraint count (|R1CS|, in thousands), proof size ($|\pi|$), prover time, and verifier time. We depict the median of five trials for prover and verifier time; variance was very small in all cases.

(and therefore right-field proving) may not be possible.

8 Implementation and Evaluation

We implemented all of this paper's sigma protocols (e.g., the sidecar protocol of §6.1) in Rust and implemented our ZKP circuits in Z#, a variant of ZoKrates language [51]. For setup, proof generation and proof verification, we utilized CirC [82], a compiler that translates Z# into R1CS. We extended CirC with set membership arguments, reverse array accesses and explicit witness computation. Our server testbed for Figure 7 is an Ubuntu 22.04 machine with a AMD Ryzen Threadripper 5995WX 1.8GHz CPU, and 256 GBs RAM. In order to get a sense of performance on less powerful hardware, we additionally tested our ECDSA-P256 and Ed25519 proofs of possession on a laptop running Arch Linux with an Intel Core Ultra 5 125H with 16GB of RAM in both a multithreaded and single-threaded setting. The results of these tests will be discussed in these schemes' respective paragraphs below.

We implemented our Dorian ZKP in Rust by extending the reference Spartan implementation. Because Dorian's costs are different than Spartan's—for example, the second sumcheck is more expensive—we optimized further with parallelism. We instantiated the underlying group with three curves: first, Spartan's default (Curve25519); second, an Arkworks [6] implementation of the T-256 curve [54]; third, a custom Rust implementation of our new T-25519 curve, based on the crrl [90] library's arithmetic mod $2^{255} - 19$. All our implementations with Groth16 or Mirage ZKP are instantiated with the BLS12-381 curve.

8.1 Evaluating Proofs of Possession

Since hashing is used for all our proofs of possession, we first evaluate our optimized SHA-256 and SHA-512 circuits in isolation, then present results for full proofs of possession.

Hashing. We benchmarked our SHA-256 implementation against the baseline from the Z# standard library

(proved with Bellman's [47] Groth16 backend). The results are in Figure 7a. There are two phenomena to discuss, related to constraint counts and prover time. The constraint counts highlight our implementation's high table setup costs (≈7k) and low per-block costs (≈6k); thus its improvement over the baseline increases with $|m|$. Prover times also improve, but not by as much. For instance, at $|m| = 2048$, our constraint counts are 77% smaller, but our prover time is only 32% smaller. This has basically two causes. First, the lookup arguments transcript (which is verified in I-R1CS) contains larger field elements than the baseline's R1CS witness. This increases the density of the multi-scalar multiplication used in Mirage/Groth16 to commit to these elements. However, state-of-the-art MSM implementations are optimized for sparse inputs in highly input-dependent ways. We further discuss the effect of density on prover time in Section 8.2. The second cause is the cost of running the lookup argument: concretely, we found that it added ≈70ms to the witness generation phase of proving for 512B messages, and increased costs elsewhere.

RSA-PKCS1v1.5. Figure 7b shows our results for proving possession of an RSA-PKCS1v1.5 (2048-bit) signature. We compared with the prior state-of-the-art implementation, (xJsnark [73]) and the most popular non-academic implementation we found [116]. Note that this implementation omits the hashing computation, biasing the comparison against us.

Our prover time was far better. For a one-block message, our prover was 3.3× faster than xJsnark, and 22.7× faster than the Circom implementation (an underestimate because it omits hashing). Even for a 2kB message, our prover takes only three seconds. In constraint counts, our system's costs for the modular exponentiation and hashing were similar for the smallest message—around 22k vs. 13k—but for the 2kB message hashing is ≈90% of the cost.

In our laptop tests, the median prover time for a 2048-byte message was 11.8 seconds when multithreaded and 20 seconds when single-threaded. Median verifier times were 62 and 35 milliseconds respectively.

ECDSA-P256. We implemented two ECDSA-P256 proofs of possession—one with right-field arithmetic (S7) over the T-256 curve, and one with all other optimizations (S5, 6). We use Dorian for right-field arithmetic, and Bellman's Mirage for our other prover. The results are in Figure 7c. We fould no peer-reviewed academic work to compare against (xJsnark implemented one P-256 benchmark, but not an ECDSA verifier). We compare against the most popular open-source project we found, in Halo2's gadget language [115]. Again, it does not hash, so the comparison is biased against us.

Both our provers perform very well and the right-field one take only 2.5 seconds for even 2kB messages. Compared with the halo2 code which we emphasize does not hash the message inside the circuit, our Dorian prover is ≈11× faster in an apples-to-apples comparison, and ≈ 5.5× faster when also hashing a 64-byte message. It does, however, have a slightly slower verifier and larger proofs. Our Mirage version is strictly better than the baselines in all metrics.

In our laptop tests, the median prover time for our Mirage implementation on a 2048-byte message was 5.0 seconds when multithreaded, while on a single thread the median prover time was 16.7 seconds. The median verifier times were 20 and 26 milliseconds, respectively. The median prover time for our Dorian implementation meanwhile was 2.8 seconds when multithreaded and 6.0 seconds when single-threaded. The median verifier times were 360 and 365 milliseconds, respectively. Though prover times between Mirage and Dorian were fairly similar on our main testbed, on this more limited hardware the Dorian prover significantly outperforms Mirage, especially when single-threaded.

Ed25519. Finally, we evaluate our two Ed25519 proofs of possession — one using right-field arithmetic (and thus using Dorian-T25519), and the other using all other optimizations (using Mirage). Here there was no other academic work, but we did find implementations in Circom and Halo2 [10,11,52]. An earlier version of this paper included comparisons against both of these implementations, however we have since found that the former contains numerous soundness issues [108] and as such is an inappropriate comparison. The Halo2 code does not hash the message at all, so these comparisons are biased against us. Results are in Figure 7d.

Here, our prover times are weaker than for other signatures, though they still surpass the Halo2 baseline. In an apples-to-apples comparison without hashing our Mirage prover (resp. Dorian) takes 0.6 (resp. 0.2) seconds, 7.5× (resp. 22.5×) faster than the Halo2 code. In an unfair comparison where our implementations additionally hash a 64-byte message our provers still outperform the Halo2 code by 4.5× and 5× respectively. Our Mirage proofs were roughly 10× smaller than the Halo2 baseline and did not increase with message size. Our Dorian proofs were much larger than the Halo2 baseline, 3× larger in the apples-to-apples comparison without hashing, and increased with the length of the message.

Our weaker prover times compared to our ECDSA-P256 implementation are primarily due to Ed25519 using SHA-512 as its hash instead of SHA-256: over 90% of the constraints of the Dorian proof are devoted to hashing—the algebraic parts of the Dorian Ed25519 verifier only cost about eight thousand constraints—and SHA-512 is a bit more than two times more expensive than SHA-256 in terms of constraints per byte hashed.

The results of our laptop tests were substantially similar to our ECDSA-P256 implementation, albeit with uniformly longer times in all tests due to the more expensive hash function. The median prover time for our Mirage implementation on a 2048-byte message was 7.7 seconds when multithreaded and 26.8 seconds when single-threaded. The median verifier times were 93 and 87 milliseconds respectively. Meanwhile, the median prover times for our Dorian implementation on this same statement were 3.8 and 7.1 seconds, respectively. The median verifier times were 191 and 233 milliseconds, respectively. Just as with ECDSA-P256, Dorian outperforms Mirage on less powerful hardware, especially in the single-threaded case.

ECDSA-based ring signatures. We compare our proof-of-possession of an ECDSA-P256 signature for a *committed* public key (S6.2) with the state-of-the-art CDLS [37]. Our prover takes 1.6 ms vs. CDLS's 502 ms; our verifier takes 1.7 ms vs. CDLS's 380 ms; and our proof is 406B vs. CDLS's 155 *k*B, achieving 312×, 215×, and 390× improvements, respectively, This huge improvement is because CDLS uses a cut-and-choose protocol that requires linear iterations in the security parameter—usually at least 128—whereas our implementation only requires 1 iteration for the same security.

Our scheme extends to a ring signature using the Groth-Kohlweiss protocol. Proof and statement sizes for a ring of size 2^n are $360n + 292$B and $33(3 + 2^n)$B, respectively. The full implementation is a work in progress.

8.2 Density

As noted in the prior subsection, some of our techniques improve constraint count more than prover time. In this subsection we discuss why: witness *density*. Informally, a witness vector is less dense when its elements are all small integers. We will see that increasing density increases zkSNARK proving time, and while our techniques do decrease constraint count, they also increase density.

Formally, we define the *density* of a field element $z \in \{0, \ldots, p-1\}$ to be $\log(z+1)/\log p$ and the density of a vector to be the mean density of its elements. Note that densities lie in the interval $[0, 1]$.

Figure 8 shows the effect of density on the runtime of SNARK provers and MSMs. MSMs, or multi-scalar multiplications, are the bottleneck operation in many SNARK provers. We show the runtime of our two SNARK provers (Mirage and Dorian) on random witnesses of varying density. We also show the runtime of Mirage's underlying MSM implementation on input field vectors of varying density. Mirage and its MSM show a gradually increasing speedup as density falls. Mirage's speedup is less than its MSM's because some of the MSMs used for Mirage have high density, regardless of the density of the Mirage witness. Dorian's speed is constant at intermediate densities, but also creeps up at very low densities.

These results explain part of the slowdown observed for our lookup-based SNARKs in the previous section. Lookups create many witness elements that are uniformly random—and therefore have high density (w.h.p.). In contrast, many of the lookup-free baseline implementations use witness values that are mostly 0 or 1—which have very low density. More concretely, we measured the witness density of our SHA-256

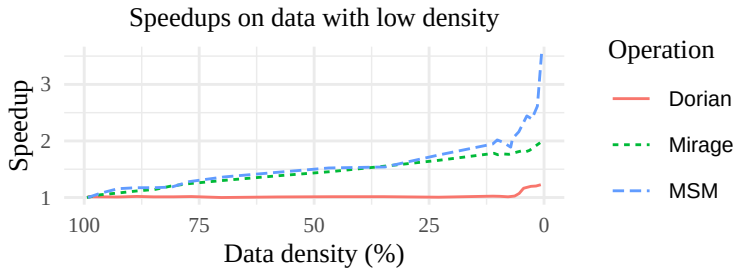

Figure 8: How the speed of SNARK provers changes with witness density. The "Dorian" and "Mirage" series shows end-to-end proving times for a random witness of varying density. The "MSM" series shows just the time of Mirage's multi-scalar multiplication—a bottleneck step in proving. All operations are faster at low densities.

implementation and the baseline implementation, for 2048B of random input data. Our implementation has witness density 52.0%, while the baseline has density 1.0%.

8.3 Summary

Overall, we believe our results are very promising: already practical in some settings, and near practicality in others. The main takeaways of our evaluation are as follows. First, while lookup arguments were effective in reducing constraint counts, our results also suggest there is more work to do to make them usable in the context of real applications, instead of just in microbenchmarks. The complexity of implementing our lookup-based circuits also suggests there are many opportunities to better integrate lookup techniques into frontends. The other main takeaway is that, at least in the context of proofs of signature possession, the algebraic parts of verification are no longer the bottleneck. Instead, the main source of overhead is computing hashes. Thus, we think future work should focus on improving the efficiency of ZKPs for legacy hashing.

9 Related Work

Existing tools. We apply existing techniques for efficient lookups in signature verifications. Lookup proofs were defined in [25], with several recent developments [5, 50, 56, 57, 92, 99, 111, 112]. We build on Haböck's construction (S2) [62].

Several proving systems [16, 40, 87] support lookups but use constraint systems different from the one we use – rank-one constraint systems (R1CS). While our techniques can be adapted to their constraint systems, one of our primary goals is to build a library for developers targeting R1CS, allowing them to easily integrate our code. We show how to encode Haböck's lookup argument into R1CS using randomized checks, but only one R1CS-based proving system—Mirage [72], a Groth16 extension—supports randomized algorithms. Mirage is incompatible with our right-field technique (S7), motivating us to design Dorian (S3).

Aside from the optimizations in zkSNARKs, prior work has explored proofs for composite statements. The seminal work [38] provides a method to combine algebraic-based proof protocols, such as Sigma protocols, with private-coin interactive proofs based on garbled circuits. Due to the use of private coins, their protocols cannot be made non-interactive. Agrawal et al. [3] designs non-interactive zero-knowledge proofs for composite statements, including proof of possesion for RSA signatures. This protocol uses a commit-and-prove zkSNARKs, enabling the proof of both algebraic statements within zkSNARKs and arithmetic representations in sigma protocols. This results in a proof containing 298 group elements, mainly due to their zkSNARKs, compared to just 4 in our approach. A direct comparison of performance is difficult because they only provide an estimate of prover performance in terms of group exponentiations, and they use

a group with an order equal to the RSA modulus, which is typically 2048 bits in practice, whereas we use a group with 255-bit order in Mirage. Applying their composite techniques to the zkSNARKs we used requires opening the commitments in the zkSNARKs circuit, which can be more computationally expensive than the RSA modular exponentiation. LegoSNARK [35] builds commit-and-prove zkSNARKs, allowing building non-interactive proof systems modularly by linking small specialized "gadget" SNARKs in a lightweight manner. However, the zkSNARKs in [3] and [35] do not support efficient lookups or randomized algorithms. Sigmabus [67] proposes a technique to move one fixed point multiplication out of the zkSNARK circuit. As discussed in S6.1, their technique is not directly applicable to our problem since they do not hide the base of the scalar multiplication, which is part of the signature in our setting. This motivates us to develop our sidecar protocol for ECDSA verification.

Our sidecar protocol uses the Elgamal cryptosystem to commit to group elements. Some threshold cryptosystems [46, 88] and threshold ECDSA protocols [36, 76] also use Elgamal for this purpose, but these protocols expose the encrypted or committed message at certain stages. Separately, some threshold ECDSA [36, 48, 49, 75, 76, 109, 110] leverage zero-knowledge proofs but expose the nonce R of the signature at some stage. In contrast, our proof of possession protocol commits to R using the Elgamal cryptosystem while ensuring that R is never revealed throughout the protocol.

Comparison to our work. Our work focuses on proof of possessions of the most widely-deployed signature schemes, including RSA PKCS1v1.5 and ECDSA-P256 with SHA256 and Ed25519 with SHA512. Cinderella [45] builds zero knowledge proofs for X.509 certificate validation, with focus on RSA verification and ASN.1 parsing, and deployed case studies for TLS and e-voting. xJsnark [73], a programming framework for verifiable computations with various optimization techniques, further improves RSA verification circuit but do not support other signature schemes. Their prover time for RSA is at least 3x compared with our work as shown in Table 7b. For verification of ECDSA over P256 and Ed25519, only [115] and [11] have reproducible implementations. Both works require one fixed-point scalar multiplication and one dynamic scalar multiplication within the circuit, each of which involves numerous costly modular reductions over the base field. The expensive dynamic scalar multiplication can be shifted outside the circuit using our sidecar technique (S6.1). Their fixed-point scalar multiplication is optimized using preprocessing techniques similar to ours. To avoid adding points using the complete formula, they leverage a random point during scalar multiplication, which assumes that the adversary does not know the discrete-logarithm relation between the random point and the base point. However, we observe that this random point is unnecessary for avoiding the complete formula when the base point's order is prime (S6), which is true for both ECDSA-P256 and Ed25519. With the same windowed size, their methods incur two additional point additions per fixed-point multiplication due to adding and subtracting the random point. Additionally, neither of their works addresses hashing. Our work outperforms theirs according to Table 7c and Table 7d.

A different setting for ECDSA verification is explored in CDLS [37] and zkAttest [54]. The setting is to prove knowledge of a signature, with public message and committed signature verification key; it is known this can be used to build ring signatures. Prior work [37, 54] provided a sigma protocol for this setting, but their proof size is larger than 100kB, making it somewhat impractical. We propose a new protocol for this setting, yielding huge improvements in the proof size and prover and verifier time.

Non-legacy proofs of possession. Several works [9, 28, 31, 33, 103] focus on proofs of possession for non-legacy signature schemes. These protocols achieve high efficiency through the specialized design of the signature schemes, but these signature schemes are not widely adopted in legacy systems. In contrast, we aim to provide compatibility with legacy signature schemes, making these protocols not directly comparable to ours. (We suspect they would have a faster prover and comparable proof size and verifier time.)

Our work can be easily adapted to build anonymous signature or anonymous credentials. Various anonymous signature schemes [7, 20, 21, 23, 41, 78, 79, 86, 93, 94, 100] are designed with specialized protocols. Our zkSNARK circuits for signature verification can be easily adapted to support "advanced" signature

schemes, such as ring signatures, group signatures, multi-signatures, and threshold signatures, by leveraging existing transformations [14] or folklore techniques. While this approach introduces some overhead compared to custom-built anonymous signatures, it offers legacy compatibility, allowing for the conversion of regular credentials into anonymous signatures without coordinating with the original signer (or, crucially, without knowledge of the signing key). It also gives more flexibility in terms of the statements that can be proved—most other works on anonymous credentials [13, 29, 44, 85, 101] only support *a priori* fixed criteria. (zk-creds [96] is a notable exception—like us, they build off of zk-SNARKs.)

We note that many of these applications would require us to use a weakly simulation extractable general-purpose zkSNARK. Neither Mirage nor Dorian are known to be (weakly) simulation extractable. Their base schemes – Groth16 and Spartan – either are, or can easily be made, weakly-simulation extractable [12, 42]. This is an interesting open question for future work.

SHA-256 in zkSNARKs. Our SHA-256 implementation is based on the Halo2 standard library's SHA-256 implementation [60], which also uses Bootle et al.'s sparse form [25]. Yet, there are two key differences. First, they target halo2 [113], a zkSNARK for Plonkish constraints—not R1CS. Second, they use 16-bit limbs and a single DS_{16} table. Instead, we use a variety of smaller tables ($\mathsf{DS}_{11}, \mathsf{DS}_{10}, \mathsf{S}_{11}, \mathsf{D}_{11}, \ldots$).

Fine-grained tables admit various cost reductions. First, their setup costs ($<$10k constraints) are *much* lower than DS_{16}'s ($\approx$66k). Second, S_b and D_b lookups have cost 1, compared to DS_b's cost of 2. This is helpful, for example, in Example 1; without the D_b tables, the total cost of the 3-ary XOR would be 12, not 9. Third, smaller tables allows us to align some shift/rotation offsets with limb boundaries, which makes those shifts/rotations free.

Acknowledgments

The authors wish to thank Joe Bonneau and the anonymous reviewers for helpful comments, and Quang Dao for a helpful discussion of our security analysis for Dorian. This research was supported by NSF grant # 2236784 and by DARPA under Agreement No. HR00112020022. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the United States Government or DARPA.

References

[1] 0xPARC. circom-ecdsa: Ecdsa implementation in circom. https://github.com/0xPARC/circom-ecdsa, 2022.

[2] Marius A Aardal and Diego F Aranha. 2DT-GLS: Faster and exception-free scalar multiplication in the GLS254 binary curve. In *ICSAC*, 2022.

[3] Shashank Agrawal, Chaya Ganesh, and Payman Mohassel. Non-interactive zero-knowledge proofs for composite statements. In *CRYPTO*, 2018.

[4] Rami Akeela and Weikeng Chen. Yafa-108/146: Implementing ed25519-embedding cocks-pinch curves in arkworks-rs. https://ia.cr/2022/1145.

[5] Héctor Masip Ardevol, Jordi Baylina Melé, Daniel Lubarov, and José L. Muñoz-Tapia. RapidUp: Multi-domain permutation protocol for lookup tables, 2022. https://ia.cr/2022/1050.

[6] arkworks contributors. `arkworks` zksnark ecosystem, 2022.

[7] Giuseppe Ateniese, Jan Camenisch, Marc Joye, and Gene Tsudik. A practical and provably secure coalition-resistant group signature scheme. In *CRYPTO*, pages 255–270. Springer, 2000.

[8] Thomas Attema, Serge Fehr, and Michael Klooß. Fiat-shamir transformation of multi-round interactive proofs. In *TCC*, 2022.

[9] Man Ho Au, Willy Susilo, and Yi Mu. Constant-size dynamic k-taa. In *Security and Cryptography for Networks: 5th International Conference, SCN 2006, Maiori, Italy, September 6-8, 2006. Proceedings 5*, pages 111–125. Springer, 2006.

[10] Ayush Shukla. Eddsa signature verification in halo2-lib. https://shuklaayu.sh/blog/axiom-ed25519/, 2024.

[11] Ayush Shukla. halo2-lib-eddsa. https://github.com/shuklaayush/halo2-lib-eddsa, 2024.

[12] Karim Baghery, Markulf Kohlweiss, Janno Siim, and Mikhail Volkhov. Another look at extraction and randomization of groth's zk-snark. In *Financial Cryptography and Data Security: 25th International Conference, FC 2021, Virtual Event, March 1–5, 2021, Revised Selected Papers, Part I 25*, pages 457–475. Springer, 2021.

[13] Foteini Baldimtsi and Anna Lysyanskaya. Anonymous credentials light. In *CCS*, pages 1087–1098, 2013.

[14] Mihir Bellare, Daniele Micciancio, and Bogdan Warinschi. Foundations of group signatures: Formal definitions, simplified requirements, and a construction based on general assumptions. In *EUROCRYPT*, pages 614–629. Springer, 2003.

[15] Mihir Bellare and Phillip Rogaway. The exact security of digital signatures-how to sign with RSA and Rabin. In *CRYPTO*, 1996.

[16] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Fast reed-solomon interactive oracle proofs of proximity. In *45th international colloquium on automata, languages, and programming (icalp 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

[17] Eli Ben Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *IEEE S&P*, 2014.

[18] Daniel J Bernstein. Curve25519: new diffie-hellman speed records. In *PKC*, 2006.

[19] Daniel J Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. High-speed high-security signatures. *Journal of cryptographic engineering*, 2012.

[20] Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures. In *CRYPTO*, pages 41–55. Springer, 2004.

[21] Dan Boneh, Manu Drijvers, and Gregory Neven. Compact multi-signatures for smaller blockchains. In *ASIACRYPT*, pages 435–464. Springer, 2018.

[22] Dan Boneh and Victor Shoup. A graduate course in applied cryptography. *Draft 0.6*, 2023.

[23] Jonathan Bootle, Andrea Cerulli, Pyrros Chaidos, Essam Ghadafi, and Jens Groth. Foundations of fully dynamic group signatures. In *ACNS*, pages 117–136. Springer, 2016.

[24] Jonathan Bootle, Andrea Cerulli, Pyrros Chaidos, Jens Groth, and Christophe Petit. Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting. In *EUROCRYPT*, 2016.

[25] Jonathan Bootle, Andrea Cerulli, Jens Groth, Sune Jakobsen, and Mary Maller. Arya: Nearly linear-time zero-knowledge proofs for correct program execution. In *ASIACRYPT*, 2018.

[26] Joppe W Bos, Craig Costello, Patrick Longa, and Michael Naehrig. Selecting elliptic curves for cryptography: an efficiency and security analysis. *Journal of Cryptographic Engineering*, 2016.

[27] Ernie Brickell, Jan Camenisch, and Liqun Chen. Direct anonymous attestation, 2004. https://ia.cr/2004/205.

[28] Jan Camenisch, Manu Drijvers, and Anja Lehmann. Anonymous attestation using the strong diffie hellman assumption revisited. In *SCN*, pages 1–20. Springer, 2016.

[29] Jan Camenisch, Maria Dubovitskaya, Kristiyan Haralambiev, and Markulf Kohlweiss. Composable and modular anonymous credentials: Definitions and practical constructions. In *ASIACRYPT*, pages 262–288. Springer, 2015.

[30] Jan Camenisch and Anna Lysyanskaya. A signature scheme with efficient protocols. In *Security in Communication Networks*, 2003.

[31] Jan Camenisch and Anna Lysyanskaya. A signature scheme with efficient protocols. In *SCN*, pages

268–289, 2003.
[32] Jan Camenisch and Anna Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In *CRYPTO*, 2004.
[33] Jan Camenisch and Anna Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In *Annual international cryptology conference*, pages 56–72. Springer, 2004.
[34] Jan Camenisch and Els Van Herreweghen. Design and implementation of the idemix anonymous credential system. In *CCS*, 2002.
[35] Matteo Campanelli, Dario Fiore, and Anaïs Querol. Legosnark: Modular design and composition of succinct zero-knowledge proofs. In *CCS*, 2019.
[36] Ran Canetti, Rosario Gennaro, Steven Goldfeder, Nikolaos Makriyannis, and Udi Peled. Uc non-interactive, proactive, threshold ecdsa with identifiable aborts. In *CCS*, pages 1769–1787, 2020.
[37] Sofia Celi, Shai Levin, and Joe Rowell. Cdls: Proving knowledge of committed discrete logarithms with soundness, 2023. https://ia.cr/2023/1595.
[38] Melissa Chase, Chaya Ganesh, and Payman Mohassel. Efficient zero-knowledge proof of algebraic and non-algebraic statements with applications to privacy preserving credentials. In *CRYPTO*, 2016.
[39] David Chaum and Eugène van Heyst. Group signatures. In *CRYPTO*, 1991.
[40] Binyi Chen, Benedikt Bünz, Dan Boneh, and Zhenfei Zhang. Hyperplonk: Plonk with linear-time prover and high-degree custom gates. In *EUROCRYPT*, pages 499–530. Springer, 2023.
[41] Lidong Chen and Torben P Pedersen. New group signature schemes. In *EUROCRYPT*, pages 171–181. Springer, 1994.
[42] Quang Dao and Paul Grubbs. Spartan and bulletproofs are simulation-extractable (for free!). In *EUROCRYPT*, 2023.
[43] Quang Dao, Jim Miller, Opal Wright, and Paul Grubbs. Weak fiat-shamir attacks on modern proof systems. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 199–216. IEEE, 2023.
[44] Alex Davidson, Ian Goldberg, Nick Sullivan, George Tankersley, and Filippo Valsorda. Privacy pass: Bypassing internet challenges anonymously. *PoPETs*, 2018.
[45] Antoine Delignat-Lavaud, Cédric Fournet, Markulf Kohlweiss, and Bryan Parno. Cinderella: Turning shabby x. 509 certificates into elegant anonymous credentials with the magic of verifiable computation. In *IEEE S&P*, 2016.
[46] Yvo Desmedt and Yair Frankel. Threshold cryptosystems. In *CRYPTO*, 1990.
[47] Zcash developers. Bellman circuit library and zksnark. https://github.com/zkcrypto/bellman.
[48] Jack Doerner, Yashvanth Kondi, Eysa Lee, and Abhi Shelat. Secure two-party threshold ecdsa from ecdsa assumptions. In *IEEE S&P*, pages 980–997. IEEE, 2018.
[49] Jack Doerner, Yashvanth Kondi, Eysa Lee, and Abhi Shelat. Threshold ecdsa from ecdsa assumptions: The multiparty case. In *IEEE S&P*, pages 1051–1066. IEEE, 2019.
[50] Liam Eagen, Dario Fiore, and Ariel Gabizon. cq: Cached quotients for fast lookups, 2022. https://ia.cr/2022/1763.
[51] Jacob Eberhardt and Stefan Tai. ZoKrates—scalable privacy-preserving off-chain computations. In *IEEE Blockchain*, 2018.
[52] Electron Labs. ed25519-circom. https://github.com/Electron-Labs/ed25519-circom, 2024.
[53] Sebastian Faust, Markulf Kohlweiss, Giorgia Azzurra Marson, and Daniele Venturi. On the non-malleability of the fiat-shamir transform. In *INDOCRYPT*, 2012.
[54] Armando Faz-Hernández, Watson Ladd, and Deepak Maram. Zkattest: Ring and group signatures for existing ecdsa keys. In *ICSAC*, 2021.
[55] Coalition for Content Provenance and Authenticity. Introducing official content credentials icon. https://c2pa.org/post/contentcredentials/.
[56] Ariel Gabizon and Dmitry Khovratovich. flookup: Fractional decomposition-based lookups in quasi-linear time independent of table size, 2022. https://ia.cr/2022/1447.

[57] Ariel Gabizon and Zachary J. Williamson. plookup: A simplified polynomial protocol for lookup tables, 2020. https://ia.cr/2020/315.
[58] Ariel Gabizon, Zachary J Williamson, and Oana Ciobotaru. Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. *Cryptology ePrint Archive*, 2019.
[59] Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. Poseidon: A new hash function for {Zero-Knowledge} proof systems. In *USENIX Security*, 2021.
[60] Jack Grigg, Daira-Emma Hopwood, and Ying Tong. 16-bit table chip for sha-256, 2022. Accessed 25/9/24.
[61] Jens Groth and Markulf Kohlweiss. One-out-of-many proofs: Or how to leak a secret and spend a coin. In *EUROCRYPT*, 2015.
[62] Ulrich Haböck. Multivariate lookups based on logarithmic derivatives, 2022. https://ia.cr/2022/1530.
[63] Tony Hansen and Donald E. Eastlake 3rd. US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF). RFC 6234, 2011.
[64] Don Johnson, Alfred Menezes, and Scott Vanstone. The elliptic curve digital signature algorithm (ecdsa). *International journal of information security*, 1:36–63, 2001.
[65] Jakob Jonsson and Burt Kaliski. Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1. RFC 3447, 2003.
[66] Simon Josefsson and Ilari Liusvaara. Edwards-Curve Digital Signature Algorithm (EdDSA). RFC 8032, January 2017.
[67] George Kadianakis, Mary Maller, and Andrija Novakovic. Sigmabus: Binding sigmas in circuits for fast curve operations, 2023. https://ia.cr/2023/1406.
[68] Burt Kaliski. PKCS #1: RSA Encryption Version 1.5. RFC 2313, March 1998.
[69] Jonathan Katz and Yehuda Lindell. *Introduction to modern cryptography: principles and protocols*. Chapman and hall/CRC, 2007.
[70] Joe Kilian and Erez Petrank. Identity escrow. In *CRYPTO*, 1998.
[71] Donald E Knuth. *The Art of Computer Programming: Seminumerical Algorithms, Volume 2*. Addison-Wesley Professional, 2014.
[72] Ahmed Kosba, Dimitrios Papadopoulos, Charalampos Papamanthou, and Dawn Song. MIRAGE: Succinct arguments for randomized algorithms with applications to universal zk-SNARKs. In *USENIX Security*, 2020.
[73] Ahmed Kosba, Charalampos Papamanthou, and Elaine Shi. xjsnark: A framework for efficient verifiable computation. In *IEEE S&P*, 2018.
[74] Information Technology Laboratory. Recommendations for discrete logarithm-based cryptography: Elliptic curve domain parameters, 2023. SP 800-186.
[75] Yehuda Lindell. Fast secure two-party ecdsa signing. In *CRYPTO*, pages 613–644. Springer, 2017.
[76] Yehuda Lindell and Ariel Nof. Fast secure multiparty ecdsa with practical distributed key generation and applications to cryptocurrency custody. In *CCS*, pages 1837–1854, 2018.
[77] Tobias Looker, Vasilis Kalos, Andrew Whitehead, and Mike Lodder. The BBS Signature Scheme. Technical report, 2023.
[78] Steve Lu, Rafail Ostrovsky, Amit Sahai, Hovav Shacham, and Brent Waters. Sequential aggregate signatures and multisignatures without random oracles. In *EUROCRYPT*, pages 465–485. Springer, 2006.
[79] Giulio Malavolta and Dominique Schröder. Efficient ring signatures in the standard model. In *ASIACRYPT*, pages 128–157. Springer, 2017.
[80] Assa Naveh and Eran Tromer. Photoproof: Cryptographic image authentication for any set of permissible transformations. In *IEEE S&P*, 2016.
[81] Tatsuaki Okamoto. Provably secure and practical identification schemes and corresponding signature

schemes. In *CRYPTO*, 1992.
[82] Alex Ozdemir, Fraser Brown, and Riad S Wahby. Circ: Compiler infrastructure for proof systems, misc verification, and more. In *IEEE S&P*, 2022.
[83] Alex Ozdemir, Evan Laufer, and Dan Boneh. Volatile and persistent memory for zkSNARKs via algebraic interactive proofs. In *IEEE S&P*, 2024.
[84] Alex Ozdemir, Riad Wahby, Barry Whitehat, and Dan Boneh. Scaling verifiable computation using efficient set accumulators. In *USENIX Security*, 2020.
[85] Christian Paquin and Greg Zaverucha. U-prove cryptographic specification v1. 1 (revision 3). *Microsoft Corporation*, pages 1–23, 2013.
[86] Sunoo Park and Adam Sealfon. It wasn't me! repudiability and claimability of ring signatures. In *CRYPTO*, pages 159–190. Springer, 2019.
[87] Luke Pearson, Joshua Fitzgerald, Héctor Masip, Marta Bellés-Muñoz, and Jose Luis Muñoz-Tapia. Plonkup: Reconciling plonk with plookup. *Cryptology ePrint Archive*, 2022.
[88] Torben Pryds Pedersen. A threshold cryptosystem without a trusted party. In *EUROCRYPT*, pages 522–526. Springer, 1991.
[89] David Pointcheval and Olivier Sanders. Short randomizable signatures. In *Topics in Cryptology-CT-RSA 2016: The Cryptographers' Track at the RSA Conference 2016, San Francisco, CA, USA, February 29-March 4, 2016, Proceedings*, pages 111–126. Springer, 2016.
[90] Thomas Pornin. crrl library, 2024.
[91] Thomas Pornin. A prime-order group with complete formulas from even-order elliptic curves. *IACR Communications in Cryptology*, 2024.
[92] Jim Posen and Assimakis A. Kattis. Caulk+: Table-independent lookup arguments, 2022. https://ia.cr/2022/957.
[93] Thomas Ristenpart and Scott Yilek. The power of proofs-of-possession: Securing multiparty signatures against rogue-key attacks. In *EUROCRYPT*, pages 228–245. Springer, 2007.
[94] Ronald L Rivest, Adi Shamir, and Yael Tauman. How to leak a secret. In *ASIACRYPT*, pages 552–565. Springer, 2001.
[95] Michael Rosenberg, Mary Maller, and Ian Miers. SNARKBlock: Federated anonymous blocklisting from hidden common input aggregate proofs, 2021. https://ia.cr/2021/1577.
[96] Michael Rosenberg, Jacob White, Christina Garman, and Ian Miers. zk-creds: Flexible anonymous credentials from zksnarks and existing identity infrastructure. In *IEEE S&P*, pages 790–808. IEEE, 2023.
[97] Karl Rubin and Alice Silverberg. Choosing the correct elliptic curve in the CM method. *Mathematics of Computation*, 2010.
[98] Srinath Setty. Spartan: Efficient and general-purpose zksnarks without trusted setup. In *CRYPTO*, 2020.
[99] Srinath Setty, Justin Thaler, and Riad Wahby. Unlocking the lookup singularity with lasso. In *EUROCRYPT*, 2024.
[100] Hovav Shacham and Brent Waters. Efficient ring signatures without random oracles. In *PKC*, pages 166–180. Springer, 2007.
[101] Alberto Sonnino, Mustafa Al-Bassam, Shehar Bano, Sarah Meiklejohn, and George Danezis. Coconut: Threshold issuance selective disclosure credentials with applications to distributed ledgers. In *NDSS*, 2018.
[102] Ernst Straus. Addition chains of vectors (problem 5125). *American Mathematical Monthly*, 70, 1964.
[103] Stefano Tessaro and Chenzhi Zhu. Revisiting bbs signatures. In *EUROCRYPT*, pages 691–721, 2023.
[104] Justin Thaler et al. Proofs, arguments, and zero-knowledge. *Foundations and Trends® in Privacy and Security*, 4(2–4):117–660, 2022.
[105] Patrick P. Tsang, Man Ho Au, Apu Kapadia, and Sean W. Smith. Blac: Revoking repeatedly

misbehaving anonymous users without relying on ttps. *ACM Trans. Inf. Syst. Secur.*, 2010.
[106] Verifiable credentials overview. https://www.w3.org/TR/2025/NOTE-vc-overview-20250210/, 2024.
[107] David Waite, Michael B. Jones, and Jeremie Miller. JSON Web Proof. Internet-Draft draft-ietf-jose-json-web-proof-07, Internet Engineering Task Force, October 2024. Work in Progress.
[108] Hongbo Wen, Jon Stephens, Yanju Chen, Kostas Ferles, Shankara Pailoor, Kyle Charbonnet, Isil Dillig, and Yu Feng. Practical security analysis of zero-knowledge proof circuits. Cryptology ePrint Archive, Paper 2023/190, 2023.
[109] Harry WH Wong, Jack PK Ma, Hoover HF Yin, and Sherman SM Chow. Real threshold ecdsa. In *NDSS*, 2023.
[110] Haiyang Xue, Man Ho Au, Xiang Xie, Tsz Hon Yuen, and Handong Cui. Efficient online-friendly two-party ecdsa signature. In *CCS*, pages 558–573, 2021.
[111] Arantxa Zapico, Vitalik Buterin, Dmitry Khovratovich, Mary Maller, Anca Nitulescu, and Mark Simkin. Caulk: Lookup arguments in sublinear time. In *CCS*, 2022.
[112] Arantxa Zapico, Ariel Gabizon, Dmitry Khovratovich, Mary Maller, and Carla Ràfols. Baloo: Nearly optimal lookup arguments, 2022. https://ia.cr/2022/1565.
[113] Zcash developers. halo2. https://github.com/zcash/halo2, 2020.
[114] Collin Zhang, Zachary DeStefano, Arasu Arun, Joseph Bonneau, Paul Grubbs, and Michael Walfish. Zombie: Middleboxes that {Don't} snoop. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*, pages 1917–1936, 2024.
[115] ZK Webauthn. webauthn-halo2. https://github.com/zkwebauthn/webauthn-halo2, 2024.
[116] zkp-application. Circom RSA verification. https://github.com/zkp-application/circom-rsa-verify, 2024.

A Security of Our Sidecar Protocol

A.1 Additional Preliminaries

A.1.1 Commitment Schemes

Definition 4 (Commitment Scheme)**.** *A commitment scheme* Com *is a pair of PPT algorithms* $(\mathsf{Setup}, \mathsf{Com})$ *with the following syntax:*

- $\mathsf{Setup}(1^\lambda) \to \mathsf{pp}$*: takes as input the security parameter and outputs public parameters* pp*,*
- $\mathsf{Com}_{\mathsf{pp}}(m, o) \to \mathsf{cm}$*: takes as input the public parameters* pp*, a message* m*, and an opening* o*, and outputs a commitment* cm*.*

For most commitment schemes, verifying if cm *is a commitment to a candidate message* m *given a public parameter* pp *involves inputting* cm*,* m *and a opening* o *and checking if* $\mathsf{cm} = \mathsf{Com}_{\mathsf{pp}}(m, o)$*.*

The commitment scheme mainly used in this work is Poseidon commitment [59]. Its public parameter is empty, so this is omitted in the later sections. We define the security properties of a commitment scheme.

Definition 5 (Hiding)**.** Com *satisfies hiding if, for any adversary* $\mathcal{A}$*, the following probability* $\mathbf{Adv}^{hide}_{\mathsf{Com}}(\mathcal{A})$ *is negligible in* λ*:*

$$\left|\Pr[\mathsf{HIDE}^{\mathcal{A}}_{0,\mathsf{Com}}(\lambda)] - \Pr[\mathsf{HIDE}^{\mathcal{A}}_{1,\mathsf{Com}}(\lambda)]\right|.$$

For computational hiding, we further restrict $\mathcal{A}$ *to be PPT. The hiding game is defined in Figure 9.*

Game $\mathsf{HIDE}_{b,\mathsf{Com}}(\lambda)$:	$\underline{LR_{\mathsf{pp},b}(m_0,m_1)}$:
$\mathsf{pp} \leftarrow\!\!\$\ \mathsf{Setup}(1^\lambda)$	$o \leftarrow\!\!\$\ \mathcal{O}$
$b' \leftarrow \mathcal{A}^{LR_{\mathsf{pp},b}(\cdot,\cdot)}(1^\lambda)$	$c \leftarrow \mathsf{Com}_{\mathsf{pp}}(m_b, o)$
return b'	return c

Figure 9: Hiding game for commitment schemes.

Definition 6 (Binding)**.** Com *satisfies binding if, for any adversary* $\mathcal{A}$*, the following probability* $\mathbf{Adv}^{bind}_{\mathsf{Com}}(\mathcal{A})$ *is negligible in* λ*:*

$$\Pr\left[\begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda) \\ (\mathsf{cm}, m_0, m_1, o_0, o_1) \leftarrow \mathcal{A}(\mathsf{pp}) \\ \mathsf{Com}_{\mathsf{pp}}(m_0, o_0) = \mathsf{Com}_{\mathsf{pp}}(m_1, o_1) \\ \wedge \quad m_0 \neq m_1 \end{array}\right].$$

For computational binding, we further restrict $\mathcal{A}$ *to be PPT.*

A.1.2 Interactive Arguments

Definition 7. *An interactive argument for a relation* $\mathcal{R}$ *is a triple of PPT algorithms* $\Pi = (\mathsf{Setup}, \mathsf{P}, V)$ *with the following syntax:*

- $\mathsf{Setup}(\mathsf{pp}_\mathsf{G}) \to \mathsf{pp}$*: takes as input the description* pp_G *of a relation* $\mathcal{R}$ *and outputs public parameters* pp*,*
- $\langle \mathsf{P}(w), V\rangle(\mathsf{pp}, \mathbb{x}) \to \{0,1\}$*: an interactive protocol whereby the prover* P*, holding a witness* w*, interacts with the verifier* V *on common input* $(\mathsf{pp}, \mathbb{x})$ *to convince* V *that* $(\mathsf{pp}, \mathbb{x}, w) \in \mathcal{R}$*. At the end,* V *outputs a bit for accept/reject.*

Definition 8 (Completeness [42])**.** *A interactive arguments is complete if for any adversary* $\mathcal{A}$*,*

$$\Pr\left[\begin{array}{c} (\mathsf{pp}, \mathbb{x}, w) \notin \mathcal{R} \vee \\ \langle \mathsf{P}(w), V\rangle(\mathsf{pp}, \mathbb{x}) = 1 \end{array} : \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Setup}(\mathsf{pp}_\mathsf{G}) \\ (\mathbb{x}, w) \leftarrow \mathcal{A}(\mathsf{pp}) \end{array}\right] = 1.$$

Definition 9 (Honest-Verifier Zero-Knowledge [53])**.** *A public-coin interactive argument* $\Pi = (\mathsf{Setup}, \mathsf{P}, V)$ *for a relation* $\mathcal{R}$ *is computational honest-verifier zero-knowledge (cHVZK) if there exists a PPT simulator* Sim *such that for all* $\mathsf{pp} \leftarrow \mathsf{Setup}(\mathsf{pp}_\mathsf{G})$ *and* $(\mathsf{pp}, \mathbb{x}, w) \in \mathcal{R}$*,*

$$\{\mathsf{View}_V\langle \mathsf{P}(\mathsf{pp}, \mathbb{x}, w), V(\mathsf{pp}, \mathbb{x})\rangle\} \approx_c \{\mathsf{Sim}(\mathsf{pp}, \mathbb{x})\}.$$

Here $\mathsf{View}_V\langle \mathsf{P}(\mathsf{pp}, \mathbb{x}, w), V(\mathsf{pp}, \mathbb{x})\rangle$ *denotes the view of the verifier, consisting of the transcript and its own randomness.*

Special soundness is defined based on trees of transcripts.

Definition 10 (Tree of Transcripts [42])**.** *Let* Π *be a* $(2r+1)$*-message public-coin interactive argument for a relation* $\mathcal{R}$*, with challenge spaces* $\mathcal{C}_1, \cdots, \mathcal{C}_r$*. Given* $\mathbf{n} = (n_1, \cdots, n_r) \in \mathbb{N}^r$ *and* $\boldsymbol{\phi} = (\phi_1, \cdots, \phi_r)$ *with* $\phi_i : \mathcal{C}_i^{n_i} \to \{0,1\}$ *for* $i \in [r]$*, we say* $\mathcal{T}$ *is a* $(\boldsymbol{\phi}, \mathbf{n})$*-tree of accepting transcripts for* pp *if:*

1. $\mathcal{T}$ *is a tree of depth* $r+1$*,*

Game $\mathsf{SS}^{\mathcal{TE},\mathcal{A}}_{\Pi,\mathcal{R},(\phi,n)}(\lambda)$:

$\mathsf{pp} \leftarrow \mathsf{Setup}(\mathsf{pp}_\mathsf{G})$
$(\mathbb{x}, \mathcal{T}) \leftarrow \mathcal{A}(\mathsf{pp})$
$w \leftarrow \mathcal{TE}(\mathsf{pp}, \mathbb{x}, \mathcal{T})$
return $(\mathsf{pp}, \mathbb{x}, w) \notin \mathcal{R} \wedge$
$\mathsf{IsAccepting}\,((\phi, n), \mathsf{pp}, \mathbb{x}, \mathcal{T})$

(a) Game for special soundness.

Game $\mathsf{KS}^{\mathcal{P}^*}_{0,\Pi}(\lambda)$:

$\mathsf{pp} \leftarrow \mathsf{Setup}(\mathsf{pp}_\mathsf{G})$
$(\mathbb{x}, \pi) \leftarrow (\mathcal{P}^*)^H(\mathsf{pp})$
$b \leftarrow V^H_{FS}(\mathsf{pp}, \mathbb{x}, \pi)$
return b

Game $\mathsf{KS}^{\mathsf{Ext},\mathcal{P}^*}_{1,\Pi,\mathcal{R}}(\lambda)$:

$\mathsf{pp} \leftarrow \mathsf{Setup}(\mathsf{pp}_\mathsf{G})$
$(\mathbb{x}, \pi) \leftarrow (\mathcal{P}^*)^H(\mathsf{pp})$
$b \leftarrow V^H_{FS}(\mathsf{pp}, \mathbb{x}, \pi)$
$w \leftarrow \mathsf{Ext}^{\mathcal{P}^*}(\mathsf{pp}, \mathbb{x}, \pi)$
return $b \wedge ((\mathsf{pp}, \mathbb{x}, w) \in \mathcal{R})$

(b) Games for knowledge soundness.

Figure 10: Games for special soundness and knowledge soundness. The extractor Ext is given black-box access to $\mathcal{P}^*$. In particular, Ext implements H for $\mathcal{P}^*$ and can rewind $\mathcal{P}^*$ to any point.

2. *For each $i \in [r+1]$, each vertex at depth i is labeled with a prover's i-th message a_i, and if $i \leq r$, has exactly n_i outgoing edges to its children, with each edge labeled with a verifier's i-th challenge $c_{i,1}, \cdots, c_{i,n_i}$ satisfying $\phi_i(c_{i,1}, \cdots, c_{i,n_i}) = 1$. Additionally, the root's label is prepended with $\mathbb{x}$ (so the label becomes $(\mathbb{x}, a_1)$),*

3. *The labels on any root-to-leaf path form a valid input-transcript pair (x, tr).*

$\mathcal{T}$ is accepting with respect to a input-transcript pair $(\mathbb{x}, tr)$ if $(\mathbb{x}, tr)$ corresponds to the left-most path of $\mathcal{T}$. Define $\mathsf{IsAccepting}\,((\boldsymbol{\phi}, \mathbf{n}), \mathsf{pp}, \mathbb{x}, (\pi,)\mathcal{T})$ to be a predicate checking whether $\mathcal{T}$ is a $(\boldsymbol{\phi}, \mathbf{n})$-tree of accepting transcripts for pp and $\mathbb{x}$, and optionally π.

In the usual definition of a tree of accepting transcripts [8, 24], ϕ_i is the *distinctness predicate*, meaning the i-th challenges $c_{i,1}, \cdots, c_{i,n_i}$ from a vertex at depth i are distinct.

Definition 11 (Special Soundness [42])**.** *Π is a $(\boldsymbol{\phi}, \mathbf{n})$-computational special sound if there exists a PPT tree-extraction algorithm $\mathcal{TE}$ such that for all EPT adversary $\mathcal{A}$, the following probability is negligible in λ:*

$$\mathbf{Adv}^{\mathsf{SS}}_{\Pi,\mathcal{R},(\boldsymbol{\phi},\mathbf{n})}(\mathcal{TE}, \mathcal{A}) := \Pr[\mathsf{SS}^{\mathcal{TE},\mathcal{A}}_{\Pi,\mathcal{R},(\boldsymbol{\phi},\mathbf{n})}(\lambda)]$$

The special soundness game is shown in Figure 10a. We say Π is computational special sound (SS) if it is $(\boldsymbol{\phi}, \mathbf{n})$-computational special sound for some $\boldsymbol{\phi}$ and $\mathbf{n}$.

Next we show that the interactive version of our sidecar protocol in Figure 4 satisfies Special Soundness and Honest-Verifier Zero-Knowledge (HVZK). We can then apply results from [42] and [53] to argue that the F-S compiled non-interactive version is adaptive knowledge-sound (Def. 2.8, [42]) and zero-knowledge (Def. 4, [53]). The completeness of our sidecar protocol is straightforward.

A.2 Adaptive Knowledge Soundness

Let ARK be the interactive version of SNARK. Let Π_{ECDSA} be the interactive version of the sidecar protocol in Figure 4 obtained by undoing all F-S transformations. We first establish special soundness (SS) of Π_{ECDSA}. We can then get knowledge soundness of its Fiat-Shamir compiled version via a standard result.

Theorem 3. *If*

1. HCom *satisfies binding, and*

2. ARK *is $(\boldsymbol{\phi}_{\mathsf{ARK}}, \mathbf{n}_{\mathsf{ARK}})$-special sound, where $\boldsymbol{\phi}_{\mathsf{ARK}} = (\phi_2, \cdots, \phi_r)$, $\mathbf{n}_{\mathsf{ARK}} = (n_2, \cdots, n_r)$,*

Relation to prove:

$$\mathcal{R}_{ECDSA} = \left\{ \begin{array}{l} (G, \mathsf{pk}; (r, R, z, m)) : \\ \mathsf{pk} = zR + (-H(m)/r)G \wedge \quad r = x(R) \mod q \quad \wedge r \neq 0 \quad \wedge z \neq 0 \end{array} \right\}$$

Relation for inner SNARK:

$$\mathcal{R}_{\mathsf{ARK}} = \left\{ \begin{array}{l} \left(K; \left(C^{(1)}, \mathsf{cm}, \mathsf{cm}', c, (s_i)_{i\in[3]}\right); \left((e_i, f_i)_{i\in[3]}, o, o', m, R\right)\right) : \\ \bigwedge_{i\in[3]} s_i = f_i + e_i c \quad \wedge \quad \mathsf{cm}' = \mathsf{HCom}\left((f_i)_{i\in[3]}, o'\right) \quad \wedge \quad \mathsf{cm} = \mathsf{HCom}\left((e_i)_{i\in[3]}, o\right) \\ \wedge \quad C^{(1)} = vK + R \quad \text{where } v = e_2 e_1^{-1} \mod q \quad \wedge \quad e_3 = H(m)/r \\ \wedge \quad r \neq 0 \quad \text{where } r = x(R) \mod q \quad \wedge \quad e_1 \neq 0 \end{array} \right\}$$

Figure 11: Relations for our sidecar protocol $\Pi^{FS}_{\mathsf{ECDSA}}$.

then Π_{ECDSA} *is* $(\boldsymbol{\phi}, \mathbf{n})$*-special sound, where*

- $\boldsymbol{\phi} = (\phi_1, \cdots, \phi_r)$ *with* ϕ_1 *as a distinctness predicate, and*
- $\mathbf{n} = (2, n_2, \cdots, n_r)$.

Concretely, let $\mathcal{TE}_{\mathsf{ARK}}$ *be an* SS *extractor for* ARK*. There exists an* SS *extractor* $\mathcal{TE}$ *for* Π_{ECDSA} *running in PPT such that, for every EPT adversary* $\mathcal{P}^*$*, there exists a EPT adversary* $\mathcal{P}^*_{\mathsf{ARK}}$ *against* SS *of* ARK *and a PPT adversary* $\mathcal{A}_{\mathsf{HCom}}$ *against binding of* HCom *such that*

$$\mathbf{Adv}^{SS}_{\Pi_{\mathsf{ECDSA}}, \mathcal{R}_{public}, (\phi, \mathbf{n})}(\mathcal{TE}, \mathcal{P}^*) \leq 2 \cdot \mathbf{Adv}^{SS}_{\mathsf{ARK}, \mathcal{R}_{\mathsf{ARK}}, (\phi_{\mathsf{ARK}}, \mathbf{n}_{\mathsf{ARK}})}(\mathcal{TE}_{\mathsf{ARK}}, \mathcal{P}^*_{\mathsf{ARK}}) + 2 \cdot \mathbf{Adv}^{binding}_{\mathsf{HCom}}(\mathcal{A}_{\mathsf{HCom}}).$$

Proof. Let $\mathcal{P}^*$ be an EPT adversary against SS of Π_{ECDSA}. Let $(\mathsf{pk}, \mathcal{T})$ be the output of $\mathcal{P}^*(\mathsf{pp})$ in the SS game. We construct the following SS extractor $\mathcal{TE}$ for Π_{ECDSA} by using the SS extractor $\mathcal{TE}_{\mathsf{ARK}}$ for ARK. $\mathcal{TE}(\mathsf{pp}, \mathsf{pk}, \mathcal{T})$:

1. Parse pp as $((G, K), \mathsf{pp}_{zk})$. Parse the root of $\mathcal{T}$ as $\left(\mathsf{pk}, (C^{(1)}, U, \mathsf{cm}, \mathsf{cm}')\right)$.
2. Parse $a_{2,1}$ of $\mathcal{T}$ as $\left((s_{i,1})_{i\in[3]}, t_1\right)$, where t_1 is the first prover message for ARK. Set $\mathcal{T}_1$ to be the left subtree of $\mathcal{T}$ with root replaced by $(\mathbb{x}_1, t_1)$, where

   $$\mathbb{x}_1 = \left(C^{(1)}, \mathsf{cm}, \mathsf{cm}', c_{1,1}, (s_{i,1})_{i\in[3]}\right).$$

   Obtain witness $y_1 = \left((e_{i,1}, f_{i,1})_{i\in[3]}, o_1, o'_1, m_1, R_1\right)$ from

   $$\mathcal{TE}_{\mathsf{ARK}}\left(\mathsf{pp}_{zk}, (C^{(1)}, \mathsf{cm}, \mathsf{cm}', c_{1,1}, (s_{i,1})_{i\in[3]}), \mathcal{T}_1\right).$$

3. Parse $a_{2,2}$ of $\mathcal{T}$ as $\left((s_{i,2})_{i\in[3]}, t_2\right)$, where t_2 is the first prover message for ARK. Set $\mathcal{T}_2$ to be the right subtree of $\mathcal{T}$ with root replaced by $(\mathbb{x}_2, t_2)$, where

   $$\mathbb{x}_2 = \left(C^{(1)}, \mathsf{cm}, \mathsf{cm}', c_{1,2}, (s_{i,2})_{i\in[3]}\right).$$

   Obtain witness $y_2 = \left((e_{i,2}, f_{i,2})_{i\in[3]}, o_2, o'_2, m_2, R_2\right)$ from

   $$\mathcal{TE}_{\mathsf{ARK}}\left(\mathsf{pp}_{zk}, (C^{(1)}, \mathsf{cm}, \mathsf{cm}', c_{1,2}, (s_{i,2})_{i\in[3]}), \mathcal{T}_2\right).$$

4. If $(e_{i,1})_{i\in[3]} \neq (e_{i,2})_{i\in[3]}$ or $(f_{i,1})_{i\in[3]} \neq (f_{i,2})_{i\in[3]}$, output $\perp$. Otherwise, output $(R_1, e_{1,1}, m_1)$.

We note that $(R_1, e_{1,1})$ is a valid ECDSA signature for the message m_1 under the public key pk if $(\mathsf{pp}, \mathsf{pk}, (R_1, e_{1,1}, m_1)) \in \mathcal{R}_{ECDSA}$ (defined in Figure 11).

The extractor $\mathcal{TE}$ we construct is PPT because $\mathcal{TE}_{\mathsf{ARK}}$ is PPT. Next, we show that the advantage of the adversary $\mathcal{P}^*$ against $\mathcal{TE}_{\mathsf{ARK}}$ in the SS game is negligible. We claim that, if $\mathcal{P}^*$ wins in the SS game of Π_{ECDSA}, then at least one of the following is true:

1. The EPT adversary $\mathcal{P}_1^*$, which outputs $\mathbb{x}_1$ and $\mathcal{T}_1$, wins the SS game of ARK;
2. The EPT adversary $\mathcal{P}_2^*$, which outputs $\mathbb{x}_2$ and $\mathcal{T}_2$, wins the SS game of ARK;
3. There exists a PPT adversary winning the binding game of HCom.

Suppose $\mathcal{P}^*$ wins in the SS game of Π_{ECDSA}. Further suppose that $\mathcal{P}_1^*$ and $\mathcal{P}_2^*$ do not win the extractor $\mathcal{TE}_{\mathsf{ARK}}$ in the SS game of ARK. We will show that there exists a PPT adversary winning the binding game of HCom.

Since $\mathcal{P}^*$ wins, we know that $(\mathsf{pk}, \mathcal{T})$ output by $\mathcal{P}^*$ satisfies $\mathsf{IsAccepting}((\boldsymbol{\phi}, \mathbf{n}), \mathsf{pp}, \mathsf{pk}, \mathcal{T})$. This means that labels on the edges of $\mathcal{T}$ satisfy the predicates $\boldsymbol{\phi}$, and the labels on any root-to-leaf path form a valid input-transcript pair (pk, tr).

We will show that $\mathsf{IsAccepting}((\boldsymbol{\phi}_{\mathsf{ARK}}, \mathbf{n}_{\mathsf{ARK}}), \mathsf{pp}_{zk}, \mathbb{x}_1, \mathcal{T}_1)$ holds. Recall that $\mathcal{T}_1$ is a subtree of $\mathcal{T}$, with its root prepended with some messages on vertices and edges leading to the subtree. Since the labels on any root-to-leaf path of $\mathcal{T}$ form a valid input-transcript pair, so do the labels on any root-to-leaf path of $\mathcal{T}_1$. Furthermore, the labels on the edges of $\mathcal{T}_1$ satisfy the predicates $\boldsymbol{\phi}_{\mathsf{ARK}}$ because the labels on the edges of $\mathcal{T}$ satisfy the predicates $\boldsymbol{\phi}$, and $\boldsymbol{\phi}_{\mathsf{ARK}}$ is the subset of $\boldsymbol{\phi}$ corresponding to predicates for the subtree. Thus, we know that $\mathsf{IsAccepting}((\boldsymbol{\phi}_{\mathsf{ARK}}, \mathbf{n}_{\mathsf{ARK}}), \mathsf{pp}_{zk}, \mathbb{x}_1, \mathcal{T}_1)$ holds. Similarly, we know that $\mathsf{IsAccepting}((\boldsymbol{\phi}_{\mathsf{ARK}}, \mathbf{n}_{\mathsf{ARK}}), \mathsf{pp}_{zk}, \mathbb{x}_2, \mathcal{T}_2)$ holds.

Therefore, we have

$$
\begin{aligned}
&\mathsf{IsAccepting}((\boldsymbol{\phi}_{\mathsf{ARK}}, \mathbf{n}_{\mathsf{ARK}}), \mathsf{pp}_{zk}, \mathbb{x}_1, \mathcal{T}_1) \quad \wedge \quad \mathsf{IsAccepting}((\boldsymbol{\phi}_{\mathsf{ARK}}, \mathbf{n}_{\mathsf{ARK}}), \mathsf{pp}_{zk}, \mathbb{x}_2, \mathcal{T}_2) \\
\implies \\
&(\mathsf{pp}_{zk}, \mathbb{x}_1, y_1) \in \mathcal{R}_{\mathsf{ARK}} \wedge (\mathsf{pp}_{zk}, \mathbb{x}_2, y_2) \in \mathcal{R}_{\mathsf{ARK}} \qquad (1) \\
&\quad \text{Note: } y_1, y_2 \text{ are extracted witnesses for } \mathcal{R}_{\mathsf{ARK}} \text{ (defined in Figure 11).} \\
\implies \\
&\mathsf{cm} = \mathsf{HCom}\left((e_{i,1})_{i\in[3]}, o_1\right) = \mathsf{HCom}\left((e_{i,2})_{i\in[3]}, o_2\right) \\
\wedge \quad &\mathsf{cm}' = \mathsf{HCom}\left((f_{i,1})_{i\in[3]}, o_1'\right) = \mathsf{HCom}\left((f_{i,2})_{i\in[3]}, o_2'\right) \qquad (2) \\
\wedge \quad &\bigwedge_{i\in[3]} s_{i,1} = f_{i,1} + e_{i,1}c_{1,1} \wedge \bigwedge_{i\in[3]} s_{i,2} = f_{i,2} + e_{i,2}c_{1,2}.
\end{aligned}
$$

Due to first two lines of Equation (2), a PPT adversary that wins the binding game of HCom exists if at least one of the following holds:

1. $(e_{i,1})_{i\in[3]} \neq (e_{i,2})_{i\in[3]} \implies$ A PPT adversary $\mathcal{A}^{(e)}_{\mathsf{HCom}}$ can use $\mathcal{TE}$ to obtain two pairs $((e_{i,1})_{i\in[3]}, o_1)$ and $((e_{i,2})_{i\in[3]}, o_2)$ to break binding of HCom.
2. $(f_{i,1})_{i\in[3]} \neq (f_{i,2})_{i\in[3]} \implies$ A PPT adversary $\mathcal{A}^{(f)}_{\mathsf{HCom}}$ can use $\mathcal{TE}$ to obtain two pairs $((f_{i,1})_{i\in[3]}, o_1')$ and $((f_{i,2})_{i\in[3]}, o_2')$ to break binding of HCom.

To show that at least one of the above holds, we will use proof by contradiction.

Suppose it were true that $(e_{i,1})_{i\in[3]} = (e_{i,2})_{i\in[3]}$ and $(f_{i,1})_{i\in[3]} = (f_{i,2})_{i\in[3]}$. Recall that $(\mathsf{pk}, \mathcal{T})$ output by $\mathcal{P}^*$ satisfies $\mathsf{IsAccepting}((\boldsymbol{\phi}, \mathbf{n}), \mathsf{pp}, \mathsf{pk}, \mathcal{T})$. Thus, we have

$$
\begin{aligned}
& s_{1,1}C^{(1)} - s_{2,1}K - s_{3,1}G = U + c_{1,1}\mathsf{pk} \quad \wedge \quad s_{1,2}C^{(1)} - s_{2,2}K - s_{3,2}G = U + c_{1,2}\mathsf{pk} \\
\Longrightarrow & \\
& \triangle s_1 C^{(1)} - \triangle s_2 K - \triangle s_3 G = \triangle c \cdot \mathsf{pk} \\
& \text{where } \triangle s_i = s_{i,1} - s_{i,2} \text{ for all } i \in [3] \text{ and } \triangle c = c_{1,1} - c_{1,2}.
\end{aligned}
$$

Since ϕ_1 is a distinctness predicate, we have $c_{1,1} \neq c_{1,2}$, implying

$$(\triangle s_1/\triangle c)C^{(1)} - (\triangle s_2/\triangle c)K - (\triangle s_3/\triangle c)G = \mathsf{pk}. \tag{3}$$

Notice in Equation (3) that at least one of $\triangle s_i$ is non-zero since pk is not an identity point. From last line of eq. (2) and the assumption that $(e_{i,1})_{i\in[3]} = (e_{i,2})_{i\in[3]}$ and $(f_{i,1})_{i\in[3]} = (f_{i,2})_{i\in[3]}$, we have, for all $i \in [3]$,

$$
\begin{aligned}
& \triangle s_i = e_{i,1}\triangle c \\
\Longrightarrow\ & e_{i,1} = \triangle s_i/\triangle c \\
\overset{eq.\ (3)}{\Longrightarrow}\ & e_{1,1}C^{(1)} - e_{2,1}K - e_{3,1}G = \mathsf{pk}.
\end{aligned} \tag{4}
$$

From Equation (1), we know that $(\mathsf{pp}_{zk}, \mathbb{x}_1, y_1) \in \mathcal{R}_{\mathsf{ARK}}$

$$
\begin{aligned}
& C^{(1)} = vK + R_1 \quad \text{where } v = e_{2,1}e_{1,1}^{-1} \mod q \quad \wedge \quad e_{3,1} = H(m_1)/r \\
& \wedge \quad x(R_1) \mod q \neq 0 \quad \wedge \quad e_{1,1} \neq 0. \\
\overset{eq.\ (4)}{\Longrightarrow} & \\
& \mathsf{pk} = e_{1,1}R_1 - (H(m_1)/r)G \quad \wedge \quad x(R_1) \mod q \neq 0 \quad \wedge \quad e_{1,1} \neq 0.
\end{aligned}
$$

This implies $(\mathsf{pp}, \mathsf{pk}, (R_1, e_{1,1}, m_1)) \in \mathcal{R}_{ECDSA}$, contradicting with the assumption that $\mathcal{P}^*$ wins. Therefore, either $(e_{i,1})_{i\in[3]} \neq (e_{i,2})_{i\in[3]}$ or $(f_{i,1})_{i\in[3]} \neq (f_{i,2})_{i\in[3]}$ holds, which implies the existence of a PPT adversary that wins the binding game of HCom.

As a result, we know that at least one of the following is true:

1. The EPT adversary $\mathcal{P}_1^*$, which outputs $\mathbb{x}_1$ and $\mathcal{T}_1$, wins the SS game of ARK;
2. The EPT adversary $\mathcal{P}_2^*$, which outputs $\mathbb{x}_2$ and $\mathcal{T}_2$, wins the SS game of ARK;
3. A PPT adversary $\mathcal{A}^{(e)}_{\mathsf{HCom}}$ can use $\mathcal{TE}$ to obtain two pairs $((e_{i,1})_{i\in[3]}, o_1)$ and $((e_{i,2})_{i\in[3]}, o_2)$ to break binding of HCom.
4. A PPT adversary $\mathcal{A}^{(f)}_{\mathsf{HCom}}$ can use $\mathcal{TE}$ to obtain two pairs $((f_{i,1})_{i\in[3]}, o_1')$ and $((f_{i,2})_{i\in[3]}, o_2')$ to break binding of HCom.

To conclude, we have

$$\mathbf{Adv}^{SS}_{\Pi_{\mathsf{ECDSA}},\mathcal{R}_{ECDSA},(\phi,\mathbf{n})}(\mathcal{TE}, \mathcal{P}^*) \leq 2\cdot\mathbf{Adv}^{SS}_{\mathsf{ARK},\mathcal{R}_{\mathsf{ARK}},(\phi_{\mathsf{ARK}},\mathbf{n}_{\mathsf{ARK}})}(\mathcal{TE}_{\mathsf{ARK}}, \mathcal{P}^*_{\mathsf{ARK}}) + 2\cdot\mathbf{Adv}^{binding}_{\mathsf{HCom}}(\mathcal{A}_{\mathsf{HCom}}).$$

Since each term in the right hand side is negligible, the advantage of $\mathcal{P}^*$ is also negligible, thereby showing that Π_{ECDSA} is $(\boldsymbol{\phi}, \mathbf{n})$-special sound. □

Theorem 4. *If Π_{ECDSA} is $(\phi, \mathbf{n})$-special sound, then its Fiat-Shamir transformed version $\Pi^{FS}_{\mathsf{ECDSA}}$ is adaptively knowledge sound.*

Proof. Let C be the minimum predicate size of ϕ. From [42], there exists an EPT extractor Ext such that for every PPT adversary P^* againt KS making at most Q random oracle calls, there exists an EPT adversary $\mathcal{A}$ against SS such that

$$\mathbf{Adv}^{KS}_{\Pi^{FS}_{\mathsf{ECDSA}},\mathcal{R}_{ECDSA}}(\mathsf{Ext}, \mathsf{P}^*) \leq \frac{Q(Q-1)/2 + (Q+1)(\sum_{i=1}^{r} n_i - r)}{C} + \mathbf{Adv}^{SS}_{\Pi_{\mathsf{ECDSA}},\mathcal{R}_{ECDSA},(\phi,\mathbf{n})}(\mathcal{TE}, \mathcal{A}).$$

Since each term in the right hand side is negligible, the advantage of P^* is also negligible, thereby showing that $\Pi^{FS}_{\mathsf{ECDSA}}$ is adaptively knowledge sound. □

Remark. For this theorem to hold, the verifier challenge in the SNARK instantiating $\Pi^{FS}_{\mathsf{ECDSA}}$ must depend on the prover's earlier messages before the SNARK proof is generated. In our implementation, this requirement is met because the SNARK's verifier challenge is derived from its statement, which includes the prover's earlier messages. However, arbitrarily combining a SNARK with an auxiliary protocol might not meet the security requirement, because, to our knowledge, existing general-purpose SNARK implementations do not support generating the verifier challenge based on an external transcript.

A.3 Zero Knowledge

Let Π_{ECDSA} be the interactive version of the sidecar protocol in Figure 4, obtained by undoing all F-S transformations except for the inner SNARK. We first establish Honest-Verifier Zero-Knowledge (HVZK) of Π_{ECDSA}. We can then get non-interactive zero-knowledge (niZK) of its Fiat-Shamir compiled version via a standard result.

Theorem 5. *The sidecar protocol Π_{ECDSA} satisfies HVZK if*

1. HCom *is statistically hiding*
2. SNARK *for the relation $\mathcal{R}_{\mathsf{ARK}}$ is niZK.*

Concretely, there exists a PPT simulator Sim for Π_{ECDSA}, constructed using a PPT simulator $\mathsf{Sim}_{\mathsf{SNARK}}$ for SNARK, such that for every PPT distinguisher $\mathcal{D}$ against Π_{ECDSA}, there exists an adversary $\mathcal{A}$ against statistically hiding of HCom, and a distinguisher $\mathcal{D}_{\mathsf{SNARK}}$ against niZK of SNARK such that, the success probability of $\mathcal{D}$ is at most

$$\mathbf{Adv}^{ZK}_{\mathsf{SNARK},\mathcal{R}_{\mathsf{SNARK}}}(\mathsf{Sim}_{\mathsf{SNARK}}, \mathcal{D}_{\mathsf{SNARK}}) + 2 \cdot \mathbf{Adv}^{hide}_{\mathsf{HCom}}(\mathcal{A}).$$

Proof. We begin by noting that the protocol we analyze here is interactive but also uses a NIZK in the third message. We implicitly assume this NIZK uses a random oracle. Thus, the verifier's view of the interactive protocol is still relative to a random oracle. This also means the NIZK simulator $\mathsf{Sim}_{\mathsf{SNARK}}$ must be given the ability to reprogram this random oracle. We will therefore refer to the simulator as $\mathsf{Sim}^{RePro}_{\mathsf{SNARK}}$. However, since the details of this do not affect our analysis, we leave the RO and the associated bookkeeping otherwise implicit in our analysis.

Consider the following HVZK simulator Sim for Π_{ECDSA}:

$\underline{\mathsf{Sim}\,(\mathsf{pp}, \mathsf{pk}):}$

1. $c \leftarrow\!\!\$\ \mathbb{F}_q$

2. $((G, K), \mathsf{pp}_{zk}) \leftarrow \mathsf{pp}$
3. $(s_i)_{i\in[3]}, (\hat{e}_i)_{i\in[3]}, (\hat{f}_i)_{i\in[3]} \leftarrow\!\!\$\ \mathbb{F}_q^3$
4. $o, o', \gamma \leftarrow\!\!\$\ \mathbb{F}_q$
5. $C^{(1)} \leftarrow \gamma K$
6. $U \leftarrow s_1 C^{(1)} - s_2 K - s_3 G - c\mathsf{pk}$
7. $\mathsf{cm} \leftarrow \mathsf{HCom}\left((\hat{e}_i)_{i\in[3]}, o\right)$
8. $\mathsf{cm}' \leftarrow \mathsf{HCom}\left((\hat{f}_i)_{i\in[3]}, o'\right)$
9. $\pi_{zk} \leftarrow \mathsf{Sim}^{RePro}_{\mathsf{SNARK}}\left(\mathsf{pp}_{zk}, \left(C^{(1)}, \mathsf{cm}, \mathsf{cm}', c, (s_i)_{i\in[3]}\right)\right)$
10. Output a simulated view consisting of $(C^{(1)}, U, \mathsf{cm}, \mathsf{cm}', (s_i)_{i\in[3]}, \pi_{zk})$ and (honest) verifier's randomness c.

Note that $\hat{f}_1, \hat{f}_2, \hat{f}_3$ might not be the representation of U in terms of $C^{(1)}$, $-K$ and $-G$, respectively, but such representation must exist since their group $E/\mathbb{F}_q$ is cyclic.

We will show that the transcript generated by Sim is computationally indistinguishable with the view of a honest verifier V running an interactive protocol Π_{ECDSA} with the prover P holding $(\mathsf{pp}, \mathsf{pk}, (R, z, m)) \in \mathcal{R}_{ECDSA}$ (defined in Figure 11).

- H_0 : The view of the honest verifier in the real protocol Π_{ECDSA} is $(C^{(1)}, U, \mathsf{cm}, \mathsf{cm}', (s_i)_{i\in[3]}, \pi_{zk})$ and the verifier's randomness c generated as follows:
  1. P parses pp as $((G, K), \mathsf{pp}_{zk})$ and generates $v, f_1, f_2, f_3 \leftarrow\!\!\$\ \mathbb{F}_q$, $e_1 \leftarrow z$, $e_2 \leftarrow z \cdot v$, $e_3 \leftarrow H(m)/r$, where $r = x(R) \mod q$. Then, P computes and sends to V the following:
     (a) $C^{(1)} \leftarrow vK + R$
     (b) $U \leftarrow f_1 C^{(1)} - f_2 K - f_3 G$
     (c) $\mathsf{cm} \leftarrow \mathsf{HCom}\left((e_i)_{i\in[3]}, o\right)$
     (d) $\mathsf{cm}' \leftarrow \mathsf{HCom}\left((f_i)_{i\in[3]}, o'\right)$.
  2. V generates and sends to P a challenge $c \leftarrow\!\!\$\ \mathbb{F}_q$.
  3. P sends to V responses $(s_i)_{i\in[3]}$ and a SNARK proof π_{zk}, where they are computed as follows:
     (a) $s_i \leftarrow f_i + e_i c$ for all $i \in [3]$
     (b) $\mathbb{x} \leftarrow \left(C^{(1)}, \mathsf{cm}, \mathsf{cm}', c, (s_i)_{i\in[3]}\right)$
     (c) $w \leftarrow \left((e_i, f_i)_{i\in[3]}, o, o', v, m, R\right)$
     (d) $\pi_{zk} \leftarrow \mathsf{SNARK.Prv}(\mathsf{pp}_{zk}, \mathbb{x}, w)$
- H_1: The view is same as H_0 except that the SNARK proof π_{zk} is generated by the simulator $\mathsf{Sim}^{RePro}_{\mathsf{SNARK}}$ for SNARK:
$$\pi_{\mathsf{SNARK}} \leftarrow \mathsf{Sim}^{RePro}_{\mathsf{SNARK}}\left(\mathsf{pp}_{zk}, \left(C^{(1)}, \mathsf{cm}, \mathsf{cm}', c, (s_i)_{i\in[3]}\right)\right)$$
- H_2: The difference from H_1 is that, in H_2, the commitment cm' is generated as follows:
  1. $\hat{f}_1, \hat{f}_2, \hat{f}_3, \hat{o'} \leftarrow\!\!\$\ \mathbb{F}_q$
  2. $\mathsf{cm}' \leftarrow \mathsf{HCom}((\hat{f}_i)_{i\in[3]}, \hat{o'})$

- H_3: The difference from H_2 is that, in H_3, the commitment cm is generated as follows:
  1. $\hat{e}_1, \hat{e}_2, \hat{e}_3, \hat{o} \leftarrow\!\!\$\ \mathbb{F}_q$
  2. $\mathsf{cm} \leftarrow \mathsf{HCom}((\hat{e}_i)_{i\in[3]}, \hat{o})$
- H_4: The view is generated by the simulator Sim. The difference from H_3 is that, in H_4:
  1. s_1, s_2, s_3 are uniformly chosen by Sim from $\mathbb{F}_q$.
  2. $C^{(1)} \leftarrow \gamma K$ for a uniformly chosen $\gamma \leftarrow\!\!\$\ \mathbb{F}_q$.
  3. Instead of computing
  $$U \leftarrow f_1 C^{(1)} - f_2 K - f_3 G,$$
  Sim computes
  $$U \leftarrow s_1 C^{(1)} - s_2 K - s_3 G - c \cdot \mathsf{pk}.$$

We will show that

$$\begin{aligned}\Delta(H_0, H_4) &\leq \sum_{i=0}^{4} \Delta(H_i, H_{i+1}) \\ &= \Delta(H_0, H_1) + \Delta(H_1, H_2) + \Delta(H_2, H_3) + \Delta(H_3, H_4) \\ &\leq \mathbf{Adv}^{ZK}_{\mathsf{SNARK},\mathcal{R}_{\mathsf{SNARK}}}(\mathsf{Sim}^{RePro}_{\mathsf{SNARK}}, \mathcal{D}_{\mathsf{SNARK}}) + 2 \cdot \mathbf{Adv}^{hide}_{\mathsf{HCom}}(\mathcal{A}).\end{aligned}$$

We will then show that

$$\begin{aligned}\Delta(H_0, H_1) &\leq \mathbf{Adv}^{ZK}_{\mathsf{SNARK},\mathcal{R}_{\mathsf{SNARK}}}(\mathsf{Sim}^{RePro}_{\mathsf{SNARK}}, \mathcal{D}_{\mathsf{SNARK}}) \\ \Delta(H_1, H_2) &\leq \mathbf{Adv}^{hide}_{\mathsf{HCom}}(\mathcal{A}) \\ \Delta(H_2, H_3) &\leq \mathbf{Adv}^{hide}_{\mathsf{HCom}}(\mathcal{A}) \\ \Delta(H_3, H_4) &= 0.\end{aligned}$$

By leveraging a distinguisher $\mathcal{D}^{(0)}$ for H_0, H_1, we construct a distingisher $\mathcal{D}_{\mathsf{SNARK}}$ against niZK of SNARK:

1. $\mathcal{D}_{\mathsf{SNARK}}$ sets pp to be $((G, K), \mathsf{pp}_{zk})$, randomly generates signing key pair $(\mathsf{pk}, \mathsf{sk})$, message $m \leftarrow\!\!\$\ \mathcal{M}$ and compute a ECDSA signature $(r, s) \leftarrow \mathsf{ECDSA.Sign}(\mathsf{sk}, m)$. $\mathcal{D}_{\mathsf{SNARK}}$ then computes $R \leftarrow (H(m)/s)G + (r/s)\mathsf{pk}$ and $z \leftarrow s/r$.
2. With $(\mathsf{pp}, \mathsf{pk}, (R, z, m))$, $\mathcal{D}_{\mathsf{SNARK}}$ generates the view by following the construction of H_0 except that π_{zk} is generated by the oracle O provided by the niZK security game. The oracle O is either $\mathsf{SNARK.Prv}(\mathsf{pp}_{zk}, \cdot, \cdot)$ or $\mathsf{SNARK.Sim}^{RePro}_{\mathsf{SNARK}}(\mathsf{pp}_{zk}, \cdot, \cdot)$, depending on whether $\mathcal{D}_{\mathsf{SNARK}}$ is in the real or simulated world.
3. This generated view is feeded to $\mathcal{D}^{(0)}$. $\mathcal{D}_{\mathsf{SNARK}}$ outputs what $\mathcal{D}^{(0)}$ returns.

Observe that $(\mathsf{pp}, \mathsf{pk}, (R, z, m))$ generated by $\mathcal{D}_{\mathsf{SNARK}}$ is in the relation $\mathcal{R}_{ECDSA}$. Combining this with the construction of $\mathcal{D}_{\mathsf{SNARK}}$, the input-witness pair input to the oracle O by $\mathcal{D}_{\mathsf{SNARK}}$ is always in the relation $\mathcal{R}_{\mathsf{SNARK}}$, and thus the view generated by $\mathcal{D}_{\mathsf{SNARK}}$ is perfect indistinguishable from

- H_0 when $O = \mathsf{SNARK.Prv}(\mathsf{pp}_{zk}, \cdot, \cdot)$
- H_1 when $O = \mathsf{SNARK.Sim}^{RePro}_{\mathsf{SNARK}}(\mathsf{pp}_{zk}, \cdot, \cdot)$

Therefore, $\Delta(H_0, H_1)$ is smaller than or equal to $\mathbf{Adv}^{ZK}_{\mathsf{SNARK},\mathcal{R}_{\mathsf{SNARK}}}(\mathsf{Sim}^{RePro}_{\mathsf{SNARK}}, \mathcal{D}_{\mathsf{SNARK}})$.

We then show that there exists an adversary $\mathcal{A}$ against hiding such that $\Delta(H_1, H_2) \leq \mathbf{Adv}^{hide}_{\mathsf{HCom}}(\mathcal{A})$ and $\Delta(H_2, H_3) \leq \mathbf{Adv}^{hide}_{\mathsf{HCom}}(\mathcal{A})$.

By leveraging a distinguisher $\mathcal{D}^{(1)}$ for H_1, H_2, we construct the following adversary $\mathcal{A}_1$ against the statistically hiding property of HCom:

1. $\mathcal{A}_1$ generates public parameter pp, a signing key pair $(\mathsf{pk}, \mathsf{sk})$, a message $m \leftarrow\!\!\$\ \mathcal{M}$ and uses the signing key to generate a ECDSA signature (R, z). $(\mathsf{pp}, \mathsf{pk}, (R, z, m)) \in \mathcal{R}_{ECDSA}$.
2. Given $(\mathsf{pp}, \mathsf{pk}, (R, z, m))$, $\mathcal{A}_1$ generates the view by following the construction in H_1 except that the commitment cm' in the view is generated by the oracle $\mathsf{HCom}.LR(\cdot, \cdot)$ provided by the statistically hiding game.
3. This generated view is feeded to $\mathcal{D}^{(1)}$, and $\mathcal{A}_1$ simply returns what $\mathcal{D}^{(1)}$ returns.

The view generated by $\mathcal{A}_1$ is perfect indistinguishable from:

- H_1 when the oracle is $\mathsf{HCom}.LR^{b=0}(\cdot, \cdot)$,
- H_2 when the oracle is $\mathsf{HCom}.LR^{b=1}(\cdot, \cdot)$.

Note that LR is defined in Figure 9.

Therefore, we have (5) $\Delta(H_1, H_2) \leq \mathbf{Adv}^{hide}_{\mathsf{HCom}}(\mathcal{A}_1)$. Similarly, by leveraging a distinguisher $\mathcal{D}^{(2)}$ for H_2, H_3, we construct the following adversary $\mathcal{A}_2$ against the statistically hiding property of HCom:

1. $\mathcal{A}_2$ generates public parameter pp, a signing key pair $(\mathsf{pk}, \mathsf{sk})$, a message $m \leftarrow\!\!\$\ \mathcal{M}$ and uses the signing key to generate a ECDSA signature (R, z).
2. Given $(\mathsf{pp}, \mathsf{pk}, (R, z, m))$, $\mathcal{A}_2$ generates the view by following the construction in H_2. The commitment cm in the view is generated by the oracle $\mathsf{HCom}.LR(\cdot, \cdot)$ provided by the statistically hiding game.
3. This generated view is feeded to $\mathcal{D}^{(2)}$, and $\mathcal{A}_2$ simply returns what $\mathcal{D}^{(2)}$ returns.

The view generated by $\mathcal{A}_2$ is perfect indistinguishable from

- H_2 when the oracle is $\mathsf{HCom}.LR^{b=0}(\cdot, \cdot)$
- H_3 when the oracle is $\mathsf{HCom}.LR^{b=1}(\cdot, \cdot)$

Therefore, we have $\Delta(H_2, H_3) \leq \mathbf{Adv}^{hide}_{\mathsf{HCom}}(\mathcal{A}_2)$. Combining this and (5), there must exist an adversary $\mathcal{A}$ against hiding such that $\Delta(H_1, H_2) \leq \mathbf{Adv}^{hide}_{\mathsf{HCom}}(\mathcal{A})$ and $\Delta(H_2, H_3) \leq \mathbf{Adv}^{hide}_{\mathsf{HCom}}(\mathcal{A})$.

We then claim that $\Delta(H_3, H_4) = 0$. The only differences between H_3 and H_4 are in $(s_i)_{i\in[3]}$, $C^{(1)}$ and U. We will show that $(s_i)_{i\in[3]}$, $C^{(1)}$ and U in H_3 is identically distributed to those in H_4. Observe that in H_3, $c, s_1, s_2, s_3, C^{(1)}$ are mutually independent, with c, s_1, s_2, s_3 both uniformly distributed over $\mathbb{F}_q$, and $C^{(1)}$ uniformly distributed in $\mathbb{G}$, which is because $C^{(1)}$ is always of the form $(v + v_r)K$ in H_3 with $v \leftarrow\!\!\$\ \mathbb{F}_q$ and some v_r satisfying $R = v_r K$. Given $c, s_1, s_2, s_3, C^{(1)}$, the value U is uniquely determined by $s_1 C^{(1)} - s_2 K - s_3 G - c\mathsf{pk}$. The above implies that $\Delta(H_3, H_4) = 0$.

As a result, we have $\Delta(H_0, H_4) \leq \mathbf{Adv}^{ZK}_{\mathsf{SNARK},\mathcal{R}_{\mathsf{SNARK}}}(\mathsf{Sim}^{RePro}_{\mathsf{SNARK}}, \mathcal{D}_{\mathsf{SNARK}}) + 2 \cdot \mathbf{Adv}^{hide}_{\mathsf{HCom}}(\mathcal{A})$, thereby showing that Π_{ECDSA} satisfies HVZK. □

Theorem 5 above, along with the fact that the first message of Π_{ECDSA} has min-entropy super-logarithmic in the security parameter, implies that its Fiat-Shamir compiled protocol $\Pi^{FS}_{\mathsf{ECDSA}}$ satisfies niZK, according to Theorem 1 of [53].

B Security of Our Ring Signature for ECDSA

Definition 12. *A ring signature is a quadruple of probabilistic polynomial time (PPT) algorithms* $\Pi = (\mathsf{Setup}, \mathsf{KGen}, \mathsf{Sign}, \mathsf{Vfy})$ *for generating a common key available to all users, generating keys for users, signing messages and verifying ring signature.*

- $\mathsf{Setup}(1^\lambda) \to \mathsf{pp}$*: Generates and outputs public parameters* pp *p available to all use*
- $\mathsf{KGen}(\mathsf{pp}) \to (\mathsf{vk}, \mathsf{sk})$*: Generates a public verification key* vk *and a private signing key* sk*.*
- $\mathsf{Sign}_{\mathsf{pp},\mathsf{sk}}(M, R) \to \sigma$*: Outputs a signature* σ *on the message* $M \in \{0,1\}^*$ *with respect to the ring* $R = (\mathsf{vk}_1, \ldots, \mathsf{vk}_N)$*. We require that* $(\mathsf{vk}, \mathsf{sk})$ *is a valid key pair output by* $\mathsf{KGen}(\mathsf{pp})$ *and that* $\mathsf{vk} \in R$*.*
- $\mathsf{Vfy}_{pp}(M, R, \sigma) \to b$*: Verifies a purported ring signature* σ *on a message* M *with respect to the ring of public keys* R*. It outputs* 1 *if accepting and* 0 *if rejecting the ring signature.*

We adapt the perfect correctness, unforgeability and anonymity properties defined in [61].

Definition 13 (Perfect Correctness [61])**.** *A ring signature scheme* $(\mathsf{Setup}, \mathsf{KGen}, \mathsf{Sign}, \mathsf{Vfy})$ *has perfect correctness if for all adversaries* $\mathcal{A}$*,*

$$\Pr\left[\begin{array}{l|l} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda); \quad (\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{KGen}(\mathsf{pp}) & \\ (M, R) \leftarrow \mathcal{A}(\mathsf{pp}, \mathsf{vk}, \mathsf{sk}); \quad \sigma \leftarrow \mathsf{Sign}_{\mathsf{pp},\mathsf{sk}}(M, R) & \mathsf{Vfy}_{\mathsf{pp}}(M, R, \sigma) = 1 \vee \mathsf{vk} \notin R \end{array}\right] = 1.$$

Definition 14 (Unforgeability [61])**.** *A ring signature scheme* $(\mathsf{Setup}, \mathsf{KGen}, \mathsf{Sign}, \mathit{Vfy})$ *is unforgeable (with respect to insider corruption) if for all PPT adversaries* $\mathcal{A}$*, the following probability is negligible in the security parameter* λ*:*

$$\Pr\left[\begin{array}{l} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda); \\ (M, R, \sigma) \leftarrow \mathcal{A}^{\mathsf{VKGen},\mathsf{Sign},\mathit{Corrupt}}(\mathsf{pp}) : \mathit{Vfy}_{\mathsf{pp}}(M, R, \sigma) = 1 \end{array}\right],$$

- VKGen *on the* i*th query picks randomness* r_i*, runs* $(\mathsf{vk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KGen}(\mathsf{pp}; r_i)$*, and returns* vk_i*.*
- $\mathsf{Sign}(i, M, R)$ *returns* $\sigma \leftarrow \mathsf{Sign}_{\mathsf{pp},\mathsf{sk}_i}(M, R)$*, provided* $(\mathsf{vk}_i, \mathsf{sk}_i)$ *has been generated by* VKGen *and* $\mathsf{vk}_i \in R$*.*
- $\mathit{Corrupt}(i)$ *returns* r_i *(from which* sk_i *can be computed), provided* $(\mathsf{vk}_i, \mathsf{sk}_i)$ *has been generated by* VKGen*.*
- $\mathcal{A}$ *outputs* (M, R, σ) *such that* Sign *has not been queried with* $(*, M, R)$ *and* R *only contains keys* vk_i *generated by* VKGen *where* i *has not been corrupted.*

Definition 15 (Anonymity [61])**.** *A ring signature scheme* $(\mathsf{Setup}, \mathsf{KGen}, \mathsf{Sign}, \mathsf{Vfy})$ *has perfect anonymity if for any PPT adversary* $\mathcal{A}$

$$\Pr\left[\begin{array}{l|l} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda); \quad (M, i_0, i_1, R) \leftarrow \mathcal{A}^{\mathsf{KGen}(\mathsf{pp})}(\mathsf{pp}) & \\ b \leftarrow \{0,1\}; \quad \sigma \leftarrow \mathsf{Sign}_{\mathsf{pp},\mathsf{sk}_{i_b}}(M, R) & \mathcal{A}(\sigma) = b \end{array}\right] \leq \frac{1}{2} + \mathrm{negl}(\lambda),$$

where $\mathcal{A}$ *chooses* i_0, i_1 *such that* $(\mathsf{vk}_{i_0}, \mathsf{sk}_{i_0}), (\mathsf{vk}_{i_1}, \mathsf{sk}_{i_1})$ *have been generated by the key generation oracle* $\mathsf{KGen}(\mathsf{pp})$ *and* $\mathsf{vk}_{i_0}, \mathsf{vk}_{i_1} \in R$*.*

Theorem 6. *The ring signature scheme Π_{Ring} (defined in Figure 12) satisfies perfect correctness. It is anonymous if the commitment scheme* Com *is hiding. It is unforgeable in the random oracle model if* Com *is computationally binding and ECDSA is unforgeable.*

Proof. Our ring signature is constructed from protocols Π_{rs} and Π_{memb} discussed in Section 6.2. The associated relations $\mathcal{R}_{rs}$ and $\mathcal{R}_{memb}$ are (re)stated in Figure 12. The full ring signature relation $\mathcal{R}_{ring}$ is also stated in that figure.

Our ring signature's security follows from the security of Π_{rs} and Π_{memb}. Let Π'_{rs} and Π'_{memb} denote the pre-Fiat-Shamir-compiled versions of Π_{rs} and Π_{memb} respectively. The protocol Π'_{rs} is a special case of the generic linear protocol defined in [22], where its completeness, (perfect) 2-special soundness and special Honest-Verifier Zero-Knowledge (sHVZK) are proven. The protocol Π'_{memb} satisfies completeness, (perfect) $(n+1)$-special soundness, and sHVZK, as established by the security proof in [61] , where n is the base-2 logarithm of the size of the ring.

Perfect correctness follows from completeness of Π_{rs} and Π_{memb}. Anonymity follows from special Honest-Verifier Zero-Knowledge of Π_{rs} and Π_{memb} and the hiding property of the commitment scheme Com, which guarantees that it is impossible to distinguish which secret key has been used to generate the ring signature. (Note that the hiding of Com relies on the discrete log assumption—concretely, being able to compute the discrete log of K to the base L allows breaking hiding. This means our scheme has only computational, and in particular non-post-quantum, anonymity.)

Now, we show that our ring signature Π_{Ring} is unforgeable in the random oracle model if the commitment scheme Com satisfies binding and ECDSA is unforgeable.

We will prove by contradiction. Suppose it were true that our ring signature Π_{Ring} is not unforgeable. Then there exists an adversary $\mathcal{A}$ having at least ϵ probability of creating a succcesful forgery, where $1/\epsilon$ is a polynomial in the security parameter λ. Let q_V, q_S, q_H be denote upper bounds on the number of queries that $\mathcal{A}$ makes to VKGen, Sign and the random oracle, respectively. These are polynomials in terms of the security parameter. We will show that $\mathcal{A}$ can be used either to construct $\mathcal{A}_{\mathsf{Com}}$ against the binding property of Com or construct $\mathcal{A}'$ against the unforgeability of ECDSA.

At a high level, we rely on the special soundness of Π_{rs} and Π_{memb}. By rewinding $\mathcal{A}$, we obtain $n+1$ successful forgeries using a specific random oracle query, allowing us to extract the witnesses for $\mathcal{R}'_{rs}$ and $\mathcal{R}_{memb}$. This either enables us to extract an ECDSA signature corresponding to a public key generated by VKGen or to construct an adversary $\mathcal{A}_{\mathsf{HCom}}$ that breaks the binding property of HCom. We replace one such public keys with the key targeted by the adversary $\mathcal{A}'$ against the unforgeability of the ECDSA signature scheme. With non-negligible probability, the extracted ECDSA signature is valid with respect to the key targeted by $\mathcal{A}'$.

Without loss of generality, assume $\mathcal{A}$ checks that it has made a successful forgery, meaning that $\mathcal{A}$ calls the random oracle on a query corresponding to the forged ring signature. Let pk be the key attacked by $\mathcal{A}'$ against the unforgeability of the ECDSA signature scheme. Whenever $\mathcal{A}$ queries VKGen, we run as in a real ring signature scheme Π_{Ring}, except on the j-th query, we return pk instead. If $\mathcal{A}$ queries $\mathsf{Sign}(j, m, \mathcal{S})$, we pick the challenge $c \leftarrow\!\!\$\ \{0,1\}^\lambda$ and use special honest verifier zero-knowledge simulators of Π_{rs} and Π_{memb} to simulate the proofs. We then program the random oracle $H(\cdot)$ such that H outputs c when inputting $\mathtt{x}$ defined to be the first-round prover messages from Π_{rs} and Π_{memb}, except if $\mathtt{x}$ has already been queried before in which case we abort (call this a type-one abort). Also, if the number of rewindings exceed $2n/\epsilon$ we abort—call this a type-two abort.

We start with running the real unforgeability experiment, i.e., instead of picking $\mathsf{vk}_j = \mathsf{pk}$, we pick $\mathsf{vk}_j = \mathsf{VKGen}(j)$ and answer all queries honestly (so we do not have type-one aborts).

Observe that an adversary that has probability γ of using a specific random oracle query in a successful forgery will be rewound $n = \gamma \cdot n/\gamma$ times on average on this query to sample n additional forgeries using this query. By Markov's Inequality, the probability of the attack entering the rewinding stage and exceeding

$2n/\epsilon$ rewindings (and thus a type-two abort) will therefore be at most $\epsilon/2$. Therefore, we have at least $\epsilon - \epsilon/2 = \epsilon/2$ chance of getting $n+1$ successful forgeries using a specific oracle query $\mathbb{x}$.

Switching to simulation of ring signatures instead of giving real ring signatures may result in a type-one abort when the simulation accidentally results in an oracle query $H(\mathbb{x})$ that has been used before, but with a different challenge. Due to the fact that the first message of Π_{memb} has min-entropy super-logarithmic in the security parameter λ, the probability of this happening is negligible (let define that as v) in λ. Recall that in each run of $\mathcal{A}$, $\mathcal{A}$ makes at most q_S signing queries, and thus a total of $q_S + q_H$ random oracle queries are made. The probability of reaching a type-one abort is at most $v \cdot (q_S + q_H) \cdot (2n/\epsilon + 1)$. Note that this term is negligible in λ since v is negligible and q_S, q_H, ϵ are polynomials in λ. Define this term as p_A.

Another problem that can arise is a collision in the $n+1$ challenges we get after rewinding. With a maximum of $q_S + q_H$ queries to the random oracle in each run of $\mathcal{A}$ we get a total risk of $\frac{p_A^2}{2 \cdot 2^\lambda}$ of having a collision in any random oracle outputs. Avoiding type-one aborts and collisions of challenges leaves us with at least $\frac{\epsilon}{2} - p_A - \frac{p_A^2}{2^{\lambda+1}}$ chance of getting $n+1$ successful forgeries using the same random oracle query with different challenges. If this happens, by $n+1$-special soundness of Π_{memb}, there exists a PPT extractor Ext_{memb} that can extract the witness (ℓ, o') for the statement $(\mathsf{cm}_{\mathsf{pk}}, \mathcal{S} = (\mathsf{pk}_0, \cdots, \mathsf{pk}_{N-1}))$ such that the following holds:

- $\ell \in \{0, \ldots, N-1\}$
- $\mathsf{cm}_{\mathsf{pk}} = \mathsf{Com}_{\hat{\mathsf{pp}}}(\mathsf{pk}_\ell, o')$.

By 2-special soundness of Π_{rs}, there exists a PPT extractor Ext_{rs} that can extract the witness (z, z', t, o) for the statement (E_0, E_1, R) such that the following holds:

- $E_0 = oL$
- $E_1 = zR + oK$
- $z'E_0 = tL$
- $z'E_1 = R + tK$.

Plugging in $E_0 = C^{(0)}$ and $E_1 = C^{(1)} + (h/r)G$ into the first two lines yields that

- $C^{(0)} = oL$
- $C^{(1)} - oK = zR - (h/r)G$.

Then for $\mathsf{pk}' = C^{(1)} - oK$, we have

- $\mathsf{cm}_{\mathsf{pk}} = \mathsf{Com}_{\hat{\mathsf{pp}}}(\mathsf{pk}', o) = (oL, \mathsf{pk}' + oK)$
- $\mathsf{pk}' = zR - (h/r)G$ where $r = x(R) \mod q$ and $h = H(m) \mod q$.

We also know that $z \neq 0$. This is because applying $E_0 = oL$ to $z'E_0 = tL$ yields $z'oL = tL$, implying $t = z'o$ since L is a generator of the group. Moreover, plugging in $t = z'o$ and $E_1 = zR + oK$ into $z'E_1 = R + tK$ yields $z'(zR + oK) = R + z'oK$, implying $z'z = 1$ in the field over which the group is defined. Therefore, $z \neq 0$. We also know that $r = x(R) \mod q \neq 0$ because R comes from a successful forgery, which requires that $r \neq 0$.

Therefore, we can get (z, R, o) such that

- $\mathsf{cm}_{\mathsf{pk}} = \mathsf{Com}_{\hat{\mathsf{pp}}}(\mathsf{pk}', o)$

- $\mathsf{pk}' = zR - (h/r)G$ where $r = x(R) \mod q$ and $h = H(m) \mod q$.
- $z \neq 0$ and $r \neq 0$.

The last two lines implies ECDSA.Vf $\left(\mathsf{pk}', (R, z), m\right)$ holds. Therefore, we have (z, R, m) such that

- $\mathsf{pk}_\ell \in \mathcal{S}$
- $\mathsf{cm}_{\mathsf{pk}} = \mathsf{Com}_{\hat{\mathsf{pp}}}(\mathsf{pk}_\ell, o') = \mathsf{Com}_{\hat{\mathsf{pp}}}(\mathsf{pk}', o)$
- ECDSA.Vf $\left(\mathsf{pk}', (R, z), m\right)$.

If $\mathsf{pk}_\ell \neq \mathsf{pk}'$, the adversary $\mathcal{A}_{\mathsf{Com}}$ knows a pair of messages and openings to break the binding property of Com. Otherwise, we have a valid ECDSA signature-message pair corresponding a verification key in $\mathcal{S}$. With $1/q_v$ probability, this key equals the key pk targeted by the adversary $\mathcal{A}'$ against the unforgeability of ECDSA. Therefore, we know that if we obtain $n + 1$ successful forgeries using the same random oracle query, then either

- $\mathcal{A}_{\mathsf{Com}}$ breaks the binding property of Com, or
- if $\mathcal{A}_{\mathsf{Com}}$ fails, then $\mathcal{A}'$ breaks the unforgeability of ECDSA with a probability at least $1/q_V$.

This implies

$$
\begin{aligned}
&\mathbf{Adv}^{binding}_{\mathsf{Com}}(\mathcal{A}_{\mathsf{Com}}) + q_v \mathbf{Adv}^{\mathsf{Sign}}(\mathcal{A}') \\
\geq\ &\Pr[\text{Obtain } n+1 \text{ successful forgeries using the same random oracle query}] \\
\geq\ &\frac{\epsilon}{2} - p_A - \frac{p_A^2}{2^{\lambda+1}}, \text{ where } p_A = v \cdot (q_S + q_H) \cdot (2n/\epsilon + 1)\,. \\
\implies &\epsilon \leq 2\mathbf{Adv}^{binding}_{\mathsf{Com}}(\mathcal{A}_{\mathsf{Com}}) + 2q_v \mathbf{Adv}^{\mathsf{Sign}}(\mathcal{A}') + 2p_A + \frac{p_A^2}{2^\lambda}.
\end{aligned}
$$

Recall that we assume that the commitment scheme Com is binding and ECDSA is unforgeable, and p_A is negligible in λ. Therefore, the right hand side is negligible, and thus the left hand side is negligible, contradicting with the assumption that $1/\epsilon$ is polynomial in λ. Therefore, our ring signature Π_{Ring} is unforgeable in the random oracle model. □

Remark. We conclude by noting that in our implementation we hash all public inputs in both Fiat-Shamir hashes, to prevent weak F-S attacks [43]. We also observe that it seems nontrivial to prove this scheme satisfies a ring-signature analogue of strong unforgeability. We believe that with the correct composition of Π_{rs} and Π_{memb}—in particular, getting the F-S hash right—our signature is strongly unforgeable, but we leave the details to future work.

Relation to prove:

$$\mathcal{R}_{ring} = \left\{ \begin{array}{l} (\mathsf{pp} = (G, \hat{\mathsf{pp}}), (\mathcal{S}, \mathsf{cm}_{\mathsf{pk}}, m, R), (z, \mathsf{pk}, o)) : \\ \quad \mathsf{pk} \in \mathcal{S} \quad \wedge \quad \mathsf{cm}_{\mathsf{pk}} = \mathsf{Com}_{\hat{\mathsf{pp}}}(\mathsf{pk}, o) \wedge \quad z \neq 0 \\ \wedge \quad \mathsf{pk} = zR - (h/r)G \quad \text{where } r = x(R) \mod q \text{ and } h = \mathbf{h}(m) \end{array} \right\}$$

Relations for sub-protocols:

$$\mathcal{R}'_{rs} = \left\{ \begin{array}{l} ((L, K), (E_0, E_1, R), (z, z', t, o)) : \\ \quad E_0 = oL \quad \wedge \quad E_1 = zR + oK \quad \wedge \quad z'E_0 = tL \quad \wedge \quad z'E_1 = R + tK \end{array} \right\}.$$

$$\mathcal{R}_{memb} = \left\{ \begin{array}{l} \big(\hat{\mathsf{pp}}, (\mathsf{cm}_{\mathsf{pk}}, \mathcal{S} = (\mathsf{pk}_0, \cdots, \mathsf{pk}_{N-1})), (\ell, o)\big) : \\ \ell \in \{0, \ldots, N-1\} \quad \wedge \quad \mathsf{cm}_{\mathsf{pk}} = \mathsf{Com}_{\hat{\mathsf{pp}}}(\mathsf{pk}_\ell, o) \end{array} \right\}$$

$\underline{\mathsf{Sign}^H_{\mathsf{pp},\mathsf{sk}}(m, \mathcal{S})}$:

$(G, L, K) \leftarrow \mathsf{pp}$
$(z, R) \leftarrow \text{ECDSA.Sign}(\mathsf{sk}, m)$
$o \leftarrow\!\!\$\ \mathbb{Z}_q$; $z' \leftarrow z^{-1} \mod q$; $t \leftarrow oz' \mod q$
$(C^{(0)}, C^{(1)}) \leftarrow \mathsf{cm}_{\mathsf{pk}}$
$r = x(R) \mod q$; $h = \mathbf{h}(m) \mod q$
$E_0 \leftarrow C^{(0)}$; $E_1 \leftarrow C^{(1)} + (h/r)G$
$\ell \leftarrow i : \mathcal{S}[i] = \mathsf{pk}$
$\pi_{rs} \leftarrow \Pi_{rs}.P^H\left((L, K); (E_0, E_1, R); (z, z', t, o)\right)$
$\pi_{memb} \leftarrow \Pi_{memb}.P^H\left((L, K); (\mathsf{cm}_{\mathsf{pk}}, \mathcal{S}); (\ell, o)\right)$
Return $\sigma = (R, (\pi_{rs}, \pi_{memb}))$

(a) The signing algorithm.

$\underline{\mathsf{Vf}^H_{\mathsf{pp}}(m, \mathcal{S}, \sigma)}$:

$(R, \pi) \leftarrow \sigma$
$(G, L, K) \leftarrow \mathsf{pp}$
$(C^{(0)}, C^{(1)}) \leftarrow \mathsf{cm}_{\mathsf{pk}}$
$E_0 \leftarrow C^{(0)}$
$r = x(R) \mod q$
$h = \mathbf{h}(m) \mod q$
$E_1 \leftarrow C^{(1)} + (h/r)G$
$(\pi_{rs}, \pi_{memb}) \leftarrow \pi$
Return $r \neq 0 \wedge \Pi_{rs}.V^H\left((L, K); (E_0, E_1, R, \pi_{rs})\right)$
$\wedge\ \Pi_{memb}.V^H\left((L, K); (\mathsf{cm}_{\mathsf{pk}}, \mathcal{S}, \pi_{memb})\right)$

(b) The verification algorithm.

Figure 12: The signing and verification algorithms for our ring signature Π_{Ring} for ECDSA. The protocols Π_{rs} and Π_{memb} and the relations $\mathcal{R}_{ring}$, $\mathcal{R}'_{rs}$ and $\mathcal{R}_{memb}$ are all as defined in Section 6.2. The commitment scheme to message M with opening o is defined as $\mathsf{Com}_{\mathsf{pp}}(M, o) = (oL, oK + M)$ where $\mathsf{pp} = (L, K)$. The function H is the hash used for Fiat-Shamir. We use $\mathbf{h}$ to denote the message hashing function used by ECDSA.