SigmaSuite:

How to Minimize Foreign Arithmetic in ZKP Circuits While Keeping Succinct Final Verification

Wyatt Benno
ICME, NovaNet
wyatt@icme.io

Abstract

Foreign field arithmetic often creates significant additional overheads in zero-knowledge proof circuits. Previous work has offloaded foreign arithmetic from proof circuits by using effective and often simple primitives such as Sigma protocols. While these successfully move the foreign field work outside of the circuit, the costs for the Sigma protocol's verifier still remains high. In use cases where the verifier is constrained computationally this poses a major challenge. One such use case would be in proof composition where foreign arithmetic causes a blowup in the costs for the verifier circuit. In this work we show that by using folding scheme with Sigmabus and other such uniform verifier offloading techniques, we can remove foreign field arithmetic from zero-knowledge proof circuits while achieving succinct final verification. We do this by applying prior techniques iteratively and accumulate the resulting verifier work into one folding proof of size $O(|F|)$ group elements, where F is the size of a single Sigma verifier's computation. Then by using an existing zkSNARK we can further compress to a proof size of $O(log\ |F|)$ which can be checked succinctly by a computationally constrained verifier.

1. Introduction

Zero-knowledge proofs (ZKPs) [GMR89] allow a prover to convince a verifier that a statement is true without revealing any additional information beyond the statement's validity. ZKPs have emerged as a powerful cryptographic primitive with numerous

applications, including blockchain scalability, privacy applications, verifying the authenticity of data, and more general verifiable computation.

Programming ZKP statements often involves formulating them as arithmetic circuits over finite fields. In the popular rank-1 constraint system (R1CS), the only permitted operations are additions and multiplications within a single finite field. Expressing these complex statements as arithmetic circuits poses significant challenges, one of which is dealing with non-native field arithmetic. Circuits sometimes require performing operations over fields that differ from the circuit's native field. One such example is performing an elliptic curve operation within an arithmetic circuit. The base elements of the curve are in a field that is different from the circuit's native field. Representing these non-native field elements inside the circuit often relies on the technique of bit-decomposition, which results in orders of magnitude increase in computational overhead.

Prior works [KMN23] [OKMZ24] have explored techniques to avoid embedding non-native arithmetic inside ZKP circuits by offloading it to an external Sigma protocol. While highly effective at removing the foreign arithmetic from the targeted arithmetic circuit, this approach has a downside in that it moves the burden to the protocol's verifier, which resides outside of the target circuit. It is often the case that a protocol's verifier needs to be computationally inexpensive. For example, when verifying on constrained blockchains or for use in proof composition [BGH19] the protocol's verifier itself can be turned into an arithmetic circuit. If done naively, Sigmabus proof composition would involve the iterative compounding of foreign field arithmetic work for the verifier.

This paper introduces SigmaSuite, a framework that minimizes the use of non-native arithmetic in a target ZKP circuit while maintaining low final verification costs. Our key insight leverages folding schemes [KST22][BC23], a primitive that efficiently aggregates ZKP statements with repeated structure. Using the properties of folding schemes we can remove foreign field work from the target circuit, while moving the compounded Sigma verifier work to the prover. Our extended protocol can be used for aggregating the additional work that would arise from iterative Sigmabus invocations or any other offloading technique that has a uniform verifier. A final succinct proof can be constructed by composing the folding proof with known zkSNARKs [KST22].

We explore future directions, such as using LatticeFold as the folding scheme for aggregating the offloaded Sigmabus verifier instances. With this it would be possible to remove all foreign field work from both the target circuit and folding verifier. This is because LatticeFold[BC24] operates entirely over a single ring R_q for both the commitment scheme and folding operations. However, in this case succinct proofs are still an active area of research.

1.1 Details of prior works Sigmabus and Folding Scheme

Sigmabus and uniform offloading techniques

Sigmabus [KMN23] is a technique that allows circuit designers to relocate expensive elliptic curve group operations outside of a zero-knowledge circuit. The key idea from is to perform a Sigma protocol outside the circuit to prove knowledge of a scalar x such that it satisfies $X = xG$. This avoids having to compute the scalar multiplication $X = xG$ directly inside the circuit which would require expensive non-native field arithmetic.

Instead, with Sigmabus the prover sends X and a binding commitment cm to x. The commitment scheme can be any binding commitment scheme, such as a circuit friendly hash function like Poseidon[GKR+21], or a polynomial commitment scheme like KZG. The goal is to prove that cm and X contain the same value x. The participants engage in a modified Schnorr protocol where the challenge c depends on both X and cm $c = Hash(X, cm, R, r_h)$. A valid Sigma protocol verifier checks that:

$$sG = R + cX$$

Where $R = rG$ and r is a random value chosen from the finite field.

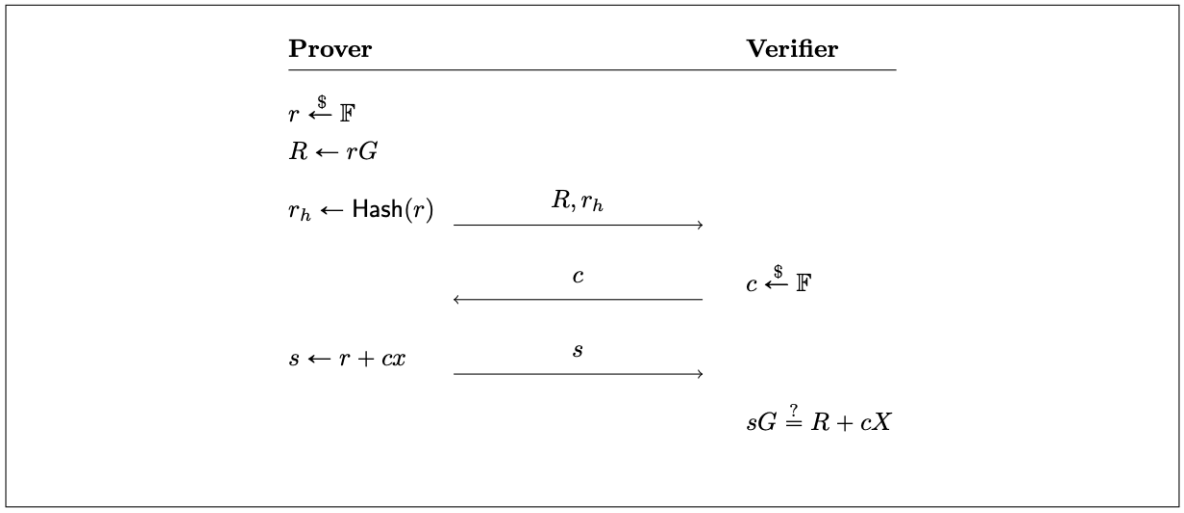

Figure 1. Modified Schnorr protocol described in [KMN23].

The target circuit '*GenZK'* is modified to accept public inputs $(X,\ cm,\ r_h,\ c,\ s)$ and witness $(x,\ r)$. The *GenZK* circuit verifies the following constraints:

$$cm = Com(x),\ r_h = Com(r),\ s = r + cx$$

A valid proof requires both the Sigma protocol and the *GenZK* proof to be valid. This allows verifying $X = xG$ without computing it directly in the target circuit. Sigmabus reduces constraints for elliptic curve operations from hundreds of thousands (in R1CS) or thousands (in Plonkish circuits) down to just a few hundred for checking SNARK-friendly hash functions and scalar field operations.

An issue arises in the proof composition setting as there could be N elliptic curve operations to tackle and therefore N Sigma protocol instances to verify. The additional work will simply grow with each step. For example, to compose N circuit invocations on the BN254 curve, we would need N group operations. If using Sigmabus naively the verifier would still need to do N group operations for the modified Sigma protocol's verifier. In the case that one wanted a succinct final verification of N instance this would be an untenable amount of work.

In later work [OKMZ24] further techniques were introduced that similarly offloaded the core work out of the arithmetic circuit and into other processes. There may be many more such 'offloading' techniques where the protocols' verifier steps are uniform across instances. We consider any resulting verifier to be uniform if the verifier's procedure is identical across multiple N instances of the offloading protocol. An example of this would be multiple instances in Sigmabus where the only difference in the Sigma protocol's verification process are the input and output values. This uniformity is what allows for the use of a uniform IVC scheme.

Folding Scheme

Nova [KST22] introduces folding schemes, a primitive that enables efficiently combining (or "folding") repeated uniform ZKP statements into a single instance. Nova also introduces the concept of 'relaxed' R1CS relations which allow for recursively folding one uniform R1CS instance into another while keeping proof size succinct. In later work, a flurry of folding schemes were introduced such as Protostar [BC23] and HyperNova [KS23a] which tackle problems of efficiency and extensions for folding other arithmetic circuit variations. LatticeFold [BC24] introduced a folding scheme outside of the discrete log setting. It uses another cryptographic primitive called Lattices which have the

additive homomorphic property needed for folding. It is interesting to note that with Lattices all work is done in the ring R_q and work with smaller finite fields than the elliptic curve protocols.

Nova extended prior works on the themes of incremental verifiable computation (IVC)[Val08][BDFG21]. Folding enables constructing better IVC where a long computation can be proven incrementally with a constant-sized proof for each step and minimal recursive overhead. This is in contrast to prior IVC schemes that rely on recursive composition of succinct arguments (like SNARKs) at each step, which is less efficient than leveraging a specialized folding scheme. At the end of the Nova folding process it is possible to take the folding proof and transform it into a succinct proof using other existing zkSNARKs [KST22]. In this way, many steps of a large incremental computation can be succinctly proven.

IVC allows the prover to take potentially non-deterministic amounts of uniform verifier work, such as multiple Sigma protocol verifiers, and prove them incrementally. IVC allows us to stop at any step and verify. This is particularly useful for long running computations. Folding based IVC not only has attractive cost and performance characteristics, but also excellent memory efficiency. A memory constrained prover could choose a step's memory footprint, scaling it up or down. A generalization of IVC called 'proof carrying data' (PCD)[CT10] allows for mutually distrustful parties to perform distributed computations that potentially run indefinitely. This could be used to parallelize the overall prover workload across multiple machines.

1.2 Our approach in a nutshell: SigmaSuite

SigmaSuite's starting point is the observation that the Sigma protocol's verifier circuit from Sigmabus itself can be transformed into an arithmetic circuit over a scalar field (e.g., a R1CS instance). We can then use this verifier circuit within folding scheme-based recursive arguments (e.g., Nova, HyperNova, ProtoStar, CycleFold), to aggregate multiple SigmaSuite Sigma verifier instances to effectively move the work to the prover. At the end of the protocol we have one modified GenZK circuit without foreign field arithmetic and one proof of folding that can be made succinct via composition with an existing zkSNARK [KST22]. By doing this, we can take many N such instances of the Sigmabus verifier circuit and aggregate them for verification into a single succinct proof.

The final cost of aggregated verification is determined by the cost of the underlying folding scheme and zkSNARK used. Moreover, many folding schemes can be rendered non-interactive via the Fiat-Shamir (FS) transform [FS87]. This means that the

offloaded work can be performed by the prover rather than the verifier. In this way, we can aggregate the cost of running many Sigmabus instances and pay a much smaller final verification cost for both:

1. The original arithmetic circuit, where foreign field constraints were removed via iterations of Sigmabus.

2. The final 'folded' verification of N aggregated Sigmabus instances.

Crucially, by instantiating the folding scheme using techniques like CycleFold [KS23b], we can represent the folding scheme's verifier circuit very efficiently on a cycle of elliptic curves. CycleFold avoids having to represent the full folding scheme verifier on the second curve, reducing it to just a few scalar multiplications. This makes the non-pairing friendly curve's component extremely small and helps to reduce the cost of verification. Moreover, it has been demonstrated that a Cyclefold proof can be used in constrained environments such as in an Ethereum smart contract verifier[1] with only ~10 million total constraints.

Overall, SigmaSuite provides an approach to minimize foreign field arithmetic in arithmetic circuits while keeping the final verifier's costs low by combining the strengths of Sigmabus for offloading expensive elliptic curve operations with folding schemes to efficiently aggregate the resulting verifier work. We envision SigmaSuite will enable efficient proof composition schemes with low overhead verifiers for use in constrained environments like blockchains and consumer devices.

1.3 Applications

We list below some applications which may benefit from these techniques.

- Removing non-native field work that arises from in circuit elliptic curve arithmetic such as multi-scalar multiplication (MSM). This often requires representing the base elements non-natively in the circuit's scalar field. In many cases this leads to a blowup in arithmetic constraints. If the prover is a high powered machine, it could be favorable to move the core work to them.

[1] https://github.com/privacy-scaling-explorations/sonobe

- Creating succinct proofs for data and assets in blockchains that use different elliptic curves.

- Dealing with non-native field work which arises during proof composition; e.g. making a proving scheme's verifier into an arithmetic circuit for use in another proving scheme. This process involves multiple instances of 'uniform verifiers' for dealing with foreign field arithmetic.

2. Technical Overview

In this section we provide an informal overview both of SigmaSuite and the security reasoning underlying it.

2.1 Overview Of The SigmaSuite Scheme

SigmaSuite builds upon the techniques introduced in Sigmabus [KMN23] for offloading expensive elliptic curve operations from arithmetic circuits. Recall that in Sigmabus, to prove a statement of knowledge of a scalar x which satisfies $X = xG$:

1. The prover sends X and a commitment cm to the scalar x.

2. They perform a modified Sigma protocol where the challenge c depends on both X and cm.

3. The prover obtains a transcript (R, c, s) proving knowledge of x such that $X = xG$.

4. The Sigma protocol's verifier checks $sG = R + cX$.

5. This transcript is incorporated into a separate arithmetic circuit that checks:
   - cm commits to x
   - $r_h = Com(r)$
   - $s = r + cx$ for a committed value r

While this avoids the expensive scalar multiplication $X = xG$ in the main circuit, Sigmabus still requires verifying N such Sigma protocol instances when composing N

elliptic curve operations. Each instance verification involves non-trivial work like checking commitments and performing a number of group operations.

SigmaSuite is used to aggregate this overhead by transforming the Sigmabus verifier circuit itself into an arithmetic circuit over a scalar field (e.g. an R1CS instance). We can then leverage folding schemes to aggregate N_i such verifier circuits, with a potentially computationally powerful prover, and transform it into a single succinct proof. The key steps are:

1. The prover generates N_i Sigma protocol transcripts (R_i, c_i, s_i) for N elliptic curve operations $X_i = x_i G$ using Sigmabus.

2. For each step, the prover modified the GenZK circuit V_i checking:
   - cm_i commits to x_i
   - $r_h = Com(r)$
   - $s_i = r_i + c_i * x_i$

3. Using a folding scheme like Nova, HyperNova, ProtoStar, or LatticeFold, the prover recursively folds the N uniform Sigma protocol verifier circuits into a single proof. This can then be made into a succinct proof 'π' using known zkSNARKs [KST22].

4. The zkSNARK verifier can efficiently check the aggregated Sigma protocol using π. And then check the GenZK circuit which no longer has any foreign field arithmetic.

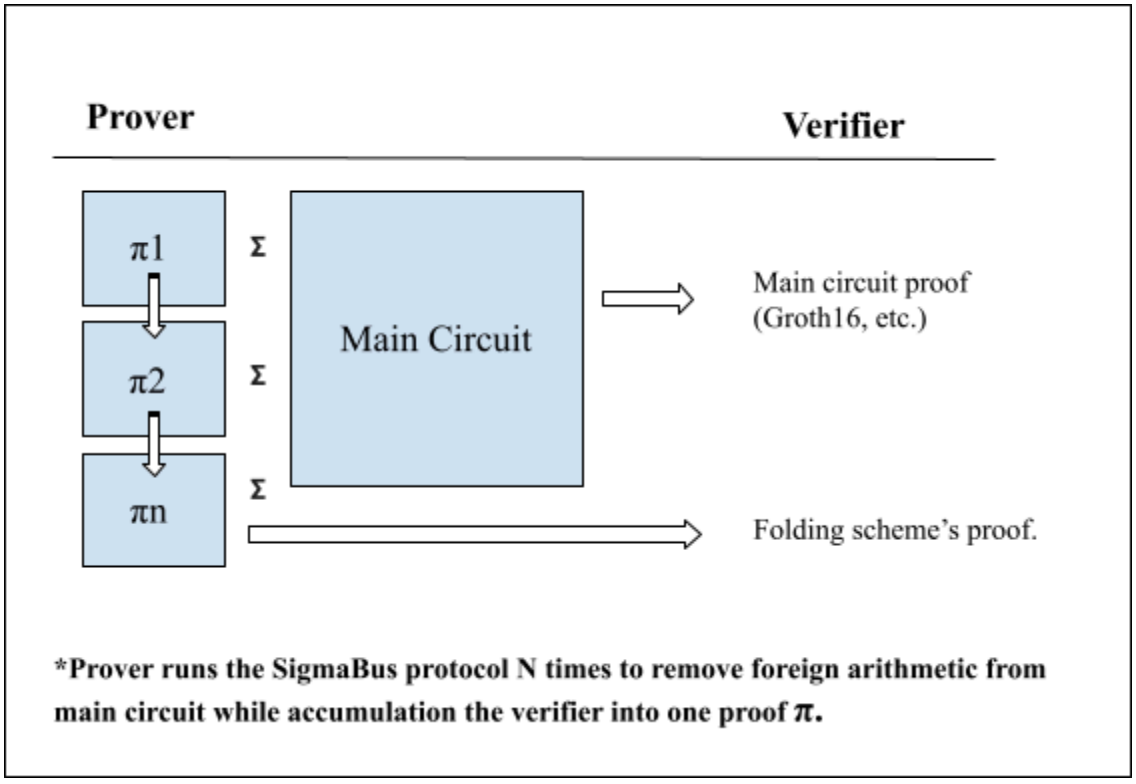

Figure 2. Layout of SigmaSuite scheme.

By instantiating the folding scheme with CycleFold [KS23b], the folding verifier is represented very efficiently over a cycle of elliptic curves, with only a few scalar multiplications on the non-pairing friendly curve. This allows highly efficient verification of the N Sigmabus instances. Moreover, a similar procedure can be followed with the uniform portions of the GenZK circuit, effectively folding all of the GenZK instances into another succinct proof.

The end result is that SigmaSuite enables minimizing non-native field arithmetic in the original circuit, while also aggregating the verification overhead into a compact proof with low-cost verification. This makes SigmaSuite well-suited for applications requiring efficient proof composition for use in constrained verifiers.

2.1 Informal Security Overview

In this section, we provide an informal intuition for why SigmaSuite satisfies the required security properties of knowledge soundness and zero-knowledge, with formal security proofs deferred to underlying primitives.

Knowledge Soundness:

The knowledge soundness property ensures that if the verifier accepts a proof, then the prover must "know" a valid witness for the statement being proven.

For SigmaSuite, knowledge soundness follows from the underlying primitives:

1. The Sigmabus protocols used for offloading each elliptic curve operation are computationally knowledge-sound, ensuring the prover knows the scalars x_i.

2. The folding scheme used to aggregate the verifier circuits is knowledge-sound, ensuring the proofs π for each V_i circuits are non-malleable.

3. The Sigmabus verifier circuit V_i itself is knowledge-sound, ensuring cm_i commits to x_i and s_i is computed correctly from x_i.

Therefore, if the verifier accepts the final aggregated proof, it implies the prover knew the witnesses x_i for all the initial elliptic curve scalar multiplications $X_i = x_i G$.

Zero-Knowledge:

The zero-knowledge property ensures the proof does not leak any information about the witness(es) beyond the truth of the statement. SigmaSuite's zero-knowledge follows from the underlying primitives being zero-knowledge:

1. The Sigmabus protocol is honest-verifier zero-knowledge (HVZK).

2. The folding scheme used for aggregation has a zero-knowledge simulator for its verifier circuit.

By combining these zero-knowledge properties, the SigmaSuite simulator can produce an untraceable proof transcript without knowing the witnesses x_i, while still being accepted by the honest verifier.

3. Costs and Future Extensions

Following is a brief description of the costs of SigmaSuite and possible future extensions for other 'uniform verifier' offloading protocols.

3.1 Costs Estimation of SigmaSuite

The final cost of verifying SigmaSuite can be broken down into two main components:

1. Cost of the target arithmetic circuit after offloading non-native operations.
2. Cost of verifying the aggregated Sigmabus instances via folding.

Cost of Original Arithmetic Circuit:

By leveraging Sigmabus to offload each non-native elliptic curve operation to a Sigma protocol, the original arithmetic circuit avoids expensive non-native field arithmetic constraints. For example, in an R1CS circuit over the BN254 scalar field, a single elliptic curve scalar multiplication would normally require thousands of constraints to represent using bit-decomposition of the BN254 base field elements. With Sigmabus, this is reduced to just a few hundred constraints in circuit to do a hash evaluation. If using Poseidon [GKR+21], the in-circuit work would roughly require 220 constraints to be expressed in R1CS. For N elliptic curve operations, Sigmabus provides an N * reduction in non-native constraints in the arithmetic circuit compared to the naive approach.

Cost of Aggregated Sigmabus Verification:

While Sigmabus avoids the N * blow-up of non-native constraints, it still requires verifying N separate Sigma protocol verifier instances, each with a number of elliptic curve operations. With SigmaSuite this work is moved to the prover running the folding protocol with final verification being succinct.

The total cost of folding depends on the choice of folding scheme used. The original Nova paper [KST22] has a stated cost of two multi-exponentiations with size $O(|F|)$. HyperNova [KS23a] reduced the prover's cost at each step to a single multi-scalar multiplication (MSM) of size equal to the number of variables in the constraint system. One can choose a folding scheme with various tradeoffs for prover speed and final verification. In Nova [KST22] with a zkSNARK wrapper the final verification time is $O(log\ |F|)$ or $O(|F|)$ depending on the commitment scheme used,

where |F| is the size of the step circuit (the uniform verifier circuit from a single instance of SigmaSuite).

By transforming the Sigma verifier circuit into an arithmetic circuit (such as R1CS) and applying a folding scheme, SigmaSuite can aggregate the verification cost of multiple instances into a single proof. This final folding proof can then be transformed into a succinct proof using known zkSNARKs [KST22]. The folding protocol can be optimized to deal with the non-native arithmetic that occurs in the folded relaxed R1CS circuits. Future optimizations could include precompiles and parallelization possible with PCD [CT10] type scheme.

3.2 Future Work and Extensions

We expect that SigmaSuite will benefit from more progress in uniform verifier offloading techniques. These could be similar to Sigma protocols or other variants that effectively remove foreign arithmetic from the target circuit. These new works can be added to SigmaSuite to extend it into an expansive tool set for dealing with foreign field arithmetic. If a succinct verifier is also created for offloading techniques the benefits would be compounded.

The efficiency of SigmaSuite heavily relies on the underlying folding scheme used for aggregating the Sigmabus verifier instances. While CycleFold [KS23b] provides an efficient instantiation, there may be opportunities to optimize folding schemes for specific scenarios or curve configurations. Future work could explore tailoring folding schemes to maximize performance in common use cases. Moreover, LatticeFold could be used to remove all foreign field work from both the target circuit and folding verifier. This would be a particularly attractive direction if succinct LatticeFold proofs are realized.

Lastly, formal security proofs and detailed analysis of SigmaSuite's knowledge soundness and zero-knowledge properties would strengthen its theoretical foundations. While the security of SigmaSuite relies on the underlying primitives, developing rigorous proofs and exploring potential optimizations within the proven secure boundaries is an important direction for future research. By pursuing these avenues, SigmaSuite can evolve into a powerful and flexible framework for tackling foreign field arithmetic in zero-knowledge arithmetic circuit, enabling efficient and scalable proof composition for a wide range of applications.

4. Acknowledgements

Special thanks to Michele Orrú, George Kadianakis, and Srinath Setty for valuable inputs and discussions.

References

[BC23] Benedikt Bunz and Binyi Chen. Protostar: Generic efficient accumulation/folding for special sound protocols. Cryptology ePrint Archive, Paper 2023/620, 2023 https://eprint.iacr.org/2023/620.

[BC24] Dan Boneh and Binyi Chen. LatticeFold: A Lattice-based Folding Scheme and its Applications to Succinct Proof Systems. Cryptology ePrint Archive, Paper 2024/257, 2024 https://eprint.iacr.org/2024/257.

[BDFG21]. Dan Boneh, Justin Drake, Ben Fisch, and Ariel Gabizon. Halo Infinite: Recursive zk-SNARKs from any Additive Polynomial Commitment Scheme. In CRYPTO, 2021.

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

[CT10] A. Chiesa and E. Tromer. “Proof-Carrying Data and Hearsay Arguments from Signature Cards”. In: Proceedings of the 1st Symposium on Innovations in Computer Science. ICS ’10. 2010, pp. 310–331.

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, CRYPTO’86, volume 263 of LNCS, pages 186–194. Springer, Heidelberg, August 1987.

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

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. SIAM Journal on Computing, 18(1):186–208, 1989.

[KMN23] George Kadianakis, Mary Maller, and Andrija Novakovic. Sigmabus: Binding Sigmas in Circuits for Fast Curve Operations. Cryptology ePrint Archive, Paper 2023/1406, 2023 https://eprint.iacr.org/2023/1406.pdf .

[KS23a] Abhiram Kothapalli and Srinath Setty. HyperNova: Recursive arguments for customizable constraint systems. Cryptology ePrint Archive, 2023 https://eprint.iacr.org/2023/573.

[KS23b] Abhiram Kothapalli and Srinath Setty. Cyclefold: Folding-scheme-based recursive arguments over a cycle of elliptic curves. Cryptology ePrint Archive, Paper 2023/1192, 2023. https://eprint.iacr.org/2023/1192.

[KST22] Abhiram Kothapalli, Srinath Setty, and Ioanna Tzialla. “Nova: Recursive Zero-Knowledge Arguments from Folding Schemes”. In: CRYPTO 2022, Part IV. Ed. by Yevgeniy Dodis and Thomas Shrimpton. Vol. 13510. LNCS. Springer, Heid

[OKMZ24] Michele Orru, George Kadianakis, Mary Maller, Greg Zaverucha. Beyond the circuit: How to Minimize Foreign Arithmetic in ZKP Circuits. Cryptology ePrint Archive, Paper 2024/265, 2024. https://eprint.iacr.org/2024/265.pdf

[VAL08] Paul Valiant.
Incrementally verifiable computation or proofs of knowledge imply time/space efficiency.
In TCC, pages 552–576, 2008.