Deliverable: Quarterly Technical Progress
14 April 2023

Prepared for:	DARPA CMO
Attention:	Mark Jones mark.jones@darpa.mil
Project Title:	*EMPHASIZE* *End-to-end Machinery for Proving* *Highly sensitive Application-oriented Statements In ZEro-knowledge*
Contract No:	HR001120C0086
SRI Project No:	P25902
Covering Period:	18 January 2023 – 14 April 2023
Principal Investigator (PI):	Stéphane Graham-Lengrand, Ph.D. Technical Director stephane.graham-lengrand@sri.com
Co-PI:	Vitor Pereira, Ph.D. Advanced Computer Scientist vitor.pereira@sri.com
Administrative contact	Colleen Conway, colleen.conway@sri.com
Address:	SRI International 333 Ravenswood Avenue, Menlo Park, CA 94025

Distribution Statement D: Distribution authorized to the Department of Defense and U.S. DoD contractors only due to the inclusion of proprietary information and to prevent Premature Dissemination of potentially critical technological information. Other requests for this document shall be referred to DARPA Public Release Center (PRC) via email.com at PRC@darpa.mil.

Quarterly Technical Progress

1 Description of Project Work in the Quarter

There are three tasks in EMPHASIZE, which according to the Statement of Work are numbered and titled as follows:

Task 4 Development of the ZK-gen platform for generating ZK proofs.
ZK-gen is EMPHASIZE's software platform for generating ZK proofs for given statements. It implements, and integrates in a unique tool, a collection of ZK proof generation mechanisms. This task focuses on development of the platform, while Task 5 develops innovative techniques and continuously integrates them to ZK-gen. Progress on Task 4 is reported in Section 1.1.

Task 5 Techniques for enhancing scalability and minimizing security assumptions.
In this task our team is developing new, innovative, techniques for enhancing scalability and minimizing security assumptions in ZK proof systems. It requires pushing the theory of ZK proof systems along 4 lines of research (subtasks 5.1-5.4). The materialization of developed techniques into the integrated software ZK-gen is performed by SRI in Task 4. Progress on Task 4 is reported in Section 1.2.

Task 6 Management and collaboration with other teams.
This task concerns the management of the EMPHASIZE team and our interactions with other SIEVE performers, i.e., agreeing on interfaces, integration, testing among other activities.

1.1 Task 4. Development of the ZK-gen platform for generating ZK proofs

1.1.1 Development of ZKgen

The ambitioned architecture of ZKgen is described in Fig. 1 and includes three main components, implementing three approaches to ZK proofs:

- The first one is a series of proof systems relying on polynomial commitments schemes (PCS);
- The second one is a series of variants of MPC-in-the-head (MitH).
- The third one is based on Line-Point Zero-Knowledge (LPZK).

ZKgen, as a TA2 software, takes the SIEVE IR2 as input. However in this architecture, the ZKgen proof systems do not ambition to *natively* support loops, switches, and function calls.

They instead rely on a converter that removes such features from IR2 and produce "IR1 simple". Native support for loops, switches, and function calls will be explored in later stages of the SIEVE program.

The PCS-based component of ZKgen is based on Plonk, which uses a custom arithmetic gate definition. We have implemented a converter from the Sieve IR to this internal plonk gate structure.

Compared to the ambitioned architecture, the current architecture of ZKgen only differs in that the three main components are separated pieces of software, each of which with its own front-end, instead of having a common front-end (i.e. a single entry point).

Use or disclosure is subject to the restrictions on the cover of this document.

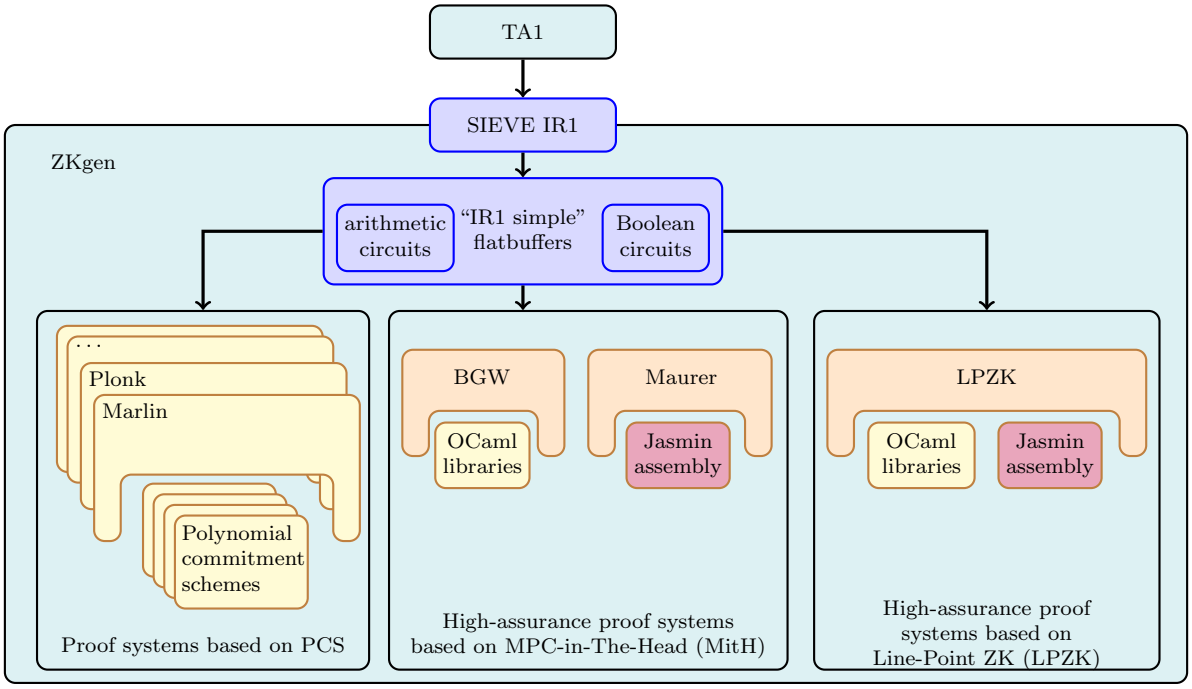


Figure 1: Ambitioned architecture of ZKgen. Yellow boxes are ZKgen components, programmed traditionally; the red box is a low-level ZKgen component, programmed traditionally and formally verified; orange boxes are ZKgen components that are automatically synthesized and formally verified by construction; blue boxes represent data in several formats; green boxes are packages; arrows show where data comes from, how it may be converted, and then fed into the ZKgen proof systems.

1.1.2 PCS-Based component

The PCS-based component of ZKgen is a STARK (Succinct Transparent ARguments of Knowledge), as it is built on top of a transparent PCS. It is developed in Rust, building on the *arkworks* ecosystem for developing and programming with zkSNARKs.[1].

We have continued the work of improving the PCS-based zkp in this quarter, which has included:

- Integrated a front-end that allows network communication between prover and verifier, as well as use of MIT LL test setup files.
- Implemented a batching optimization, where multiple points can be evaluated on one or many polynomial simultaneously.
- Implemented a run-time substitution optimization that effectively make copy-gates zero-cost.
- Updated to a recent version of zkInterface, which allows us to support the new sieve-ir.
- Updated the implementation of the arkworks BN-254 curve, allowing us to run sieve benchmarks on the new plonk implementation.
- Upgraded the parallel framework previously used with FRI (crossbeam [2]) to one with more advanced parallelization features (rayon [3]).
- Analyzed deliverable performance on the MIT LL test infrastructure.

[1] https://github.com/arkworks-rs
[2] https://github.com/crossbeam-rs/crossbeam
[3] https://github.com/rayon-rs/rayon

Use or disclosure is subject to the restrictions on the cover of this document.

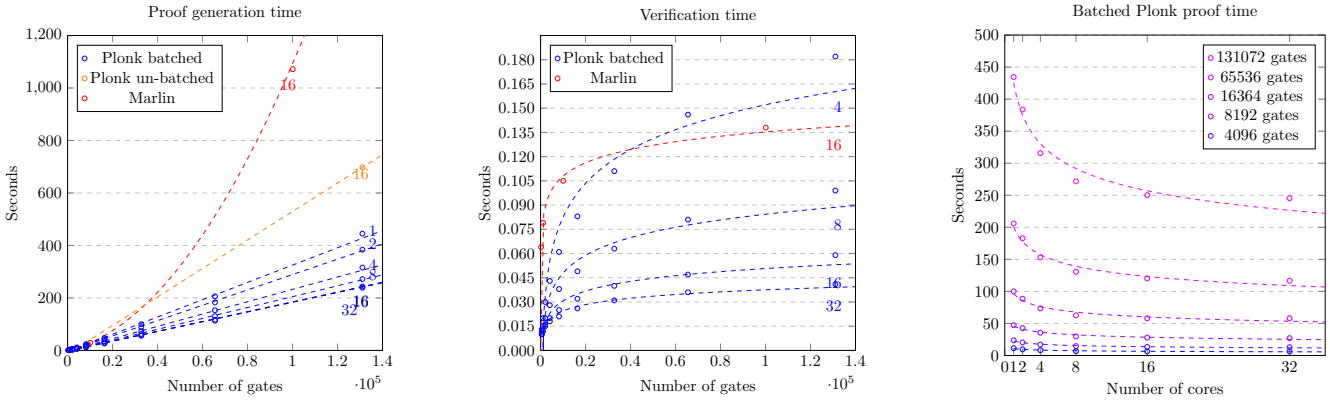


Figure 2: Performance of the batched Plonk implementation (in blue), the phase 1 Marlin implementation (in red), and the un-batched Plonk implementation (in orange). Note that the numbers next to the lines of best fit represent the number of cores used in that run.

The PCS-based proof system generates a non-interactive proof, meaning that the prover runs and generates a proof file, which can then be loaded into the verifier, and verified. This gives many advantages, such as proof portability and lack of required real-time interaction, however the testing infrastructure requires the prover and the verifier to interact via a network protocol defined by a json file, not via files saved locally or hardcoded locations. We implemented a front-end for the proof system which reads in the sieve files and test setup specifier, and handles the network interaction between the prover and verifier systems.

Previously, we had used a thread-safe data-structure library crossbeam in order to parallelize FRI. We updated FRI to use a data-parallelization library rayon instead, that uses a scheduling algorithm to more optimally distribute work over available cores than most manual implementations.In addition, polynomial commitment scheme-based proof systems can leverage batching, where multiple points on one or more polynomials can be evaluated simultaneously, giving significant performance improvements. In addition, we implemented a copy-gate optimization (a form of the copy propagation optimization), which results in copy gates having zero proof-time overhead. We also leveraged a batched version of FRI which, when combined with out updated parallelization framework, reduced the runtime of Plonk to approximately a third of the April 2022 version of Plonk.

We also conducted a performance analysis to determine the performance of the phase 2 deliverable, and compare it to the eariler un-optimized version of Plonk and Marlin. The performance increases gained by these optimizations can be clearly seen in figure 2, consisting of selected graphs of the performance analysis. A more complete analysis is submitted in the 4.4 task report, but it is immediately apparent that our updated PCS is much more performant than the previous implementations.

1.1.3 Computer-aided Verification and Synthesis of ZK Protocols and Software

Our work in the last quarter was centered in the submission, rebuttal and revision of the CCS'23 paper entitled *Boosting the Performance of High-Assurance Cryptography: Parallel Execution and Optimizing Memory Access in Formally-Verified Line-Point Zero-Knowledge* describing the contributions that resulted from the collaboration thread with the WIZKIT team. The focus of this collaboration was the formal verification of the Line-Point Zero Knowledge (LPZK) protocol and the secure optimization of the resulting high-assurance implementation.

The paper was assessed as requiring *minor revision*, as we were asked to further clarify some formalization and theoretic details, as well as to provide more performance insights, including a clear comparison

Use or disclosure is subject to the restrictions on the cover of this document.

		MM32				MM64				MM128			
		PRV	VER	TOT	TPG	PRV	VER	TOT	TPG	PRV	VER	TOT	TPG
Sequential	$2^{30} - 2^{18} + 1$	26	28	54	1.7	236	268	504	1.9	1538	1496	3034	1.4
	$2^{61} - 1$	41	35	76	2.3	342	308	650	2.5	2194	1797	3991	2
	$2^{255} - 19$	156	148	304	9.2	1261	1169	2430	9.2	9985	10463	20449	9.7
32 cores	$2^{30} - 2^{18} + 1$	11	12	23	0.7	95	82	177	0.7	830	605	1435	0.7
	$2^{61} - 1$	15	13	28	0.8	123	94	216	0.8	982	690	1672	0.8
	$2^{255} - 19$	53	40	93	2.8	408	254	661	2.5	3498	2224	5722	2.7
	#gates	59378 (32768)				535766 (262144)				4243456 (2097152)			

Table 1: Performance analysis the array-based verified LPZK implementation. Each test instance is identified by *MMX*, with *X* being the size of the matrix side. *PRV* represents the prover total time, *VER* represents the verifier total time, *TOT* represents the total protocol time and *TPG* represents the time-per-gate with respect to multiplications. All times are given in milliseconds, except the TPG, which is given in microseconds. The last row of the table portrays the total gates of the matrix multiplication circuit, with the number multiplications being shown between parentheses.

against the manual LPZK implementation by WIZKIT and against other NIZK protocols. The minor revision is now concluded.

Moreover, we also dedicated a significant amount of time to the March 10 software deliverable for the testing event. Our effort was mostly concentrated in adopting the wiztoolkit framework as our parsing front-end and on testing our software deliverable to minimize potential implementation errors.

In this document, we will report on the final performance numbers that were not yet reported on the previous quarterly report. These include:

- execution times of our deliverable, depicting the different performance of the different protocol phases
- comparison against manual LPZK implementation from WIZKIT
- comparison against manual implementations of other NIZK protocols

Benchmarking Methodology.

All measurements were conducted on an Amazon Web Services (AWS) EC2 instance, with an Intel Xeon Scalable processor clocked at 3.1 GHz, with 32 virtual cores (16 physical cores with two threads) and with 32 GB RAM capacity. We test our implementations against a matrix multiplication circuit, where we consider matrices of size 2^n, with incremental n. We also run our tests considering three different field sizes, namely $2^{30} - 2^{18} + 1$ (32 bit prime), $2^{61} - 1$ (64 bit prime) and $2^{255} - 19$ (256 bit prime), and use a 10 hours timeout.

For each test, we collect execution times of the prover and of the verifier, which include the time taken to compute the commit and to generate the commitment message (for the prover) and the time taken to parse the received message and decide the proof aceptance (for the verifier). From them, we derive the total protocol time and the *time-per-gate*. We measure the time-per-gate considering only the number of multiplication gates, which we obtain by 2^{n^3}. All measured times were obtained programatically via dedicated OCaml system calls.

Matrix multiplication with various primes: sequential vs parallel.

Table 1 presents the performance of our array-based LPZK implementation for various prime fields. We report the times of the sequential version and of the 32-core parallel version, that aggregates both the parallelism and array optimizations, giving the fastest times achieved by our verified LPZK implementation. We collect performance times over the three finite fields that we are considering. This benchmark used fully flat matrix multiplication circuits obtained from IR1 descriptions.

Indeed, changing to an array-based memory management reduces the overall complexity of the implementation, from a quadratic one to a linear one, as exposed by the constant time-per-gate measurement

Use or disclosure is subject to the restrictions on the cover of this document.

Optimized $\mathbb{F}_{2^{61}-1}$ backend

	IT-LPZKv1				IT-LPZKv2				ROM-LPZKv2				VERIFIED IT-LPZKv1			
	PRV	VER	TOT	TPG	PRV	VER	TOT	TPG	PRV	VER	TOT	TPG	PRV	VER	TOT	TPG
$2^{61}-1$	23	16	39	0.04	18	12	30	0.03	234	213	447	0.4	1274	1051	2325	2.3

Generic field backend

	IT-LPZKv1				IT-LPZKv2				ROM-LPZKv2				VERIFIED IT-LPZKv1			
	PRV	VER	TOT	TPG	PRV	VER	TOT	TPG	PRV	VER	TOT	TPG	PRV	VER	TOT	TPG
$2^{61}-1$	1611	1120	2731	2.7	1304	900	2204	2.2	2340	1295	3635	3.6	1274	1051	2325	2.3
$2^{255}-19$	1976	1525	3501	3.5	1648	1176	2824	2.8	2172	1088	3260	3.3	4912	4472	9384	9.4

Table 2: Comparison between the high-assurance sequential array-based implementation of LPZK against the manual implementation of [DILO22], as measured by executing a circuit with one million multiplication gates. The upper table shows a comparison with the manual LPZK implementation using an optimized $\mathbb{F}_{2^{61}-1}$ backend, while the lower table shows a comparison with the manual LPZK implementation using a generic NTL based field backend. All times are given in milliseconds, except the TPG, which is given in microseconds

that is achieved in the array-based implementation. Consequently, the code execution times dropped considerably.

Comparison against unverified LPZK implementation.

For a fairer comparison, we will resort to the sequential array-based implementation when comparing to existing implementations of other ZK protocols, since most of them do not incorporate parallelism mechanisms. We start by giving a comparison with the the unverified implementation of LPZK given in [DILO22], sketched in Table 2, where we compare the two implementations against a circuit with a million multiplication gates. For the manual LPZK implementation, we measured the times of IT-LPZKv1, IT-LPZKv2 and ROM-LPZKv2. The unverified LPZK implementation is instantiated first with a highly optimized backend for the $\mathbb{F}_{2^{61}-1}$ field and then with a generic finite field backend based on the NTL library, to provide a fair comparison with the verified LPZK implementation obtained from the EasyCrypt where field operations are done using OCaml bindings to the GMP library.

When comparing against the unverified LPZK implementation that uses the optimized $\mathbb{F}_{2^{61}-1}$ field, the verified LPZK implementation is 70x slower, which is reflected in the 2 orders of magnitude that separate the TPG measurement in both. This difference is amplified if one compares against the manual IT-LPZKv2 implementation. However, when the field operations are done using a generic field backend, we can conclude that our verified implementation is actually slightly faster that the unverified one. Concretely, we can observe 1.3x performance gains in the prover, verifier and total protocol times. Even considering IT-LPZKv2, the performance of the verified LPZK implementation achieves similar performance, even being slightly faster in terms of prover execution time. This contrasts with the performance measured for the $2^{255}-19$ field, where the unverified LPZK implementation is actually 2.7x faster than the verified one. Since both implementations do not make algorithmic changes with respect to the finite field, these discrepancies are justified by the different encodings made by the NTL and GMP libraries.

Comparison against unverified state-of-the-art NIZK protocols.

Taking advantage of Table 3 of [DILO22], we can also compare our LPZK implementation against other state-of-the-art NIZK protocols and zkSNARKs, namely Groth16 [Gro16], Virgo [ZXZS19], Cerberus [LSTW21] and QuickSilver [YSWW21], that we summarize in Table 3. Such table provides the execution times of the afore mentioned protocols when evaluating a circuit with one million gates over $\mathbb{F}_{2^{61}-1}$, except Cerberus which is executed over a generic 128-bit field. All measurements were done on

Use or disclosure is subject to the restrictions on the cover of this document.

Groth16				Virgo				Cerberus				QuickSilver				Verified LPZK			
PRV	VER	TOT	TPG	PRV	VER	TOT	TPG	PRV	VER	TOT	TPG	PRV	VER	TOT	TPG	PRV	VER	TOT	TPG
21k	2	21k	21	478	12	490	0.5	21.7k	148	22k	2	128	128	256	0.3	1274	1051	2325	2.3

Table 3: Comparison between the high-assurance sequential array-based implementation of LPZK against other state-of-the-art NIZK protocols, as measured by executing a circuit with one million multiplication gates. The protocols are all executed over $\mathbb{F}_{2^{61}-1}$ except Cerberus, which is executed over a generic 128-bit field. All times are given in milliseconds, except the TPG, which is given in microseconds

top of an AWS machine with equal specifications to the one used to collect the benchmarks of our verified LPZK implementation. The values report put our sequential array-based LPZK implementation with better prover performance than Groth16 and Cerberus, and with equal performance when comparing to Virgo. Nevertheless, those implementations achieve better verification times, also due to the nature of the protocols.

We also compare our code against other ZK codebases, based on MPC-in-the-Head, concretely that of ZKBoo [GMO16], that also makes use of parallelism. The authors report a 13ms prover time and a 5ms verifier time for the evaluation of a fine-tuned SHA-1 circuit with 40 AND gates, 372 XOR gates and 325 addition gates, that combines operations over $\mathbb{F}_{2^{32}}$ and over $\mathbb{F}_2$, using an 4 GHz 8 core machine. Under the same core setting, our LPZK implementation is able to evaluate a similar boolean circuit in 2.5ms (1.3ms prover time and 1.2ms verifier time). However, doing a single run of LPZK for a boolean circuit would yield a soundness error of $\frac{1}{2}$, thus requiring the performance of a series of protocol repetitions to achieve a negligible soundness error. Concretely, to achieve a 2^{-80} soundness error as the experiences performed by ZKBoo, our implementation would have an estimate execution time of 104ms for the prover and 96ms, a 8x penalty compared with the time reported by ZKBoo. Note, nevertheless, that our benchmarking machine uses a 1Ghz slower processing cores.

1.2 Task 5. Techniques for enhancing scalability and minimizing security assumptions

1.2.1 Stanford

Hyperplonk.

Plonk is a widely used succinct non-interactive proof system that uses univariate polynomial commitments. Plonk is quite flexible: it supports circuits with low-degree "custom" gates as well as circuits with lookup gates (a lookup gates ensures that its input is contained in a predefined table). For large circuits, the bottleneck in generating a Plonk proof is the need for computing a large FFT.

In Eurocrypt 2023 we presented HyperPlonk [CBBZ23], a fast adaptation of Plonk to the boolean hypercube, using multilinear polynomial commitments. HyperPlonk retains the flexibility of Plonk, but provides a number of additional benefits. First, it avoids the need for an FFT during proof generation. Second, and more importantly, it supports custom gates of much higher degree than Plonk, without harming the running time of the prover. Both of these can dramatically speed-up the prover's running time. Since HyperPlonk relies on multilinear polynomial commitments, we revisited two elegant constructions: one from Orion and one from Virgo. We showed how to reduce the Orion opening proof size to less than 10kb (an almost factor 1000 improvement), and showed how to make the Virgo FRI-based opening proof simpler and shorter.

zkPi: Proving Lean Theorems in Zero-Knowledge.

We submitted a paper [OLB23] to IEEE S&P 2023 showing how a prover can convince a verifier that a Lean theorem is true, without revealing anything else about the theorem. The core technical innovation is a new SNARK for proving dependent types, the central component needed to verify Lean proofs. We evaluate our system on theorems from the two Lean libraries: `stdlib` and `mathlib`. We can generate

Use or disclosure is subject to the restrictions on the cover of this document.

a ZK proof for 43% and 9% of theorems in these libraries, respectively, within 3 minutes. The main application for this system is the ability to prove that a secret program satisfies its specification, without revealing anything else about the program. Our submission to IEEE S&P 2023 is likely to be accepted after we revise the paper.

Using ZK to fight disinformation.

At the Real World Crypto 2023 conference we showed how to use non-interactive succinct zero-knowledge proofs to fight disinformation in the media. Our system enables a browser to verify the metadata (location and time) associated with an image in a news article. Our technique builds on the C2PA standard for content provenance. The system enables a newspaper to crop, resize, and grayscale the images taken on a C2PA-compliant camera, while retaining the ability of a browser to verify the provenance of the provided edited image. More information is available at [DB22].

ZK Circuit Verifier.

This work [OKTB23] studies satisfiability modulo the theory of finite fields and gives a decision procedure for this theory. We implemented our procedure for prime fields inside the cvc5 SMT solver. Using this theory, we constructed SMT queries that verify the correctness of various zero knowledge proof compilers on a number of input programs. Our experiments show that our implementation is vastly superior to previous approaches which encode field arithmetic using integers or bit-vectors.

Arithmetic sketching.

The Prio system, and several other previous works, achieve lightweight, low-communication zero knowledge verification of secret-shared vectors. We submitted a paper to Crypto 2023 [BBC+23] that studies the concept of an arithmetic sketch — a generalization of the core primitive used in all these systems. In addition to the formalization of an arithmetic sketch, our contributions are:

- A general framework for constructing arithmetic sketch schemes from algebraic varieties. This framework unifies schemes from prior work and gives rise to schemes for useful new languages with improved soundness error.
- The first arithmetic sketch schemes for languages of sparse vectors: vectors with bounded Hamming weight, bounded L_1 norm, and vectors whose few non-zero values satisfy a given predicate.
- A method for "compiling" any arithmetic sketch scheme for a language $\mathcal{L}$ into a low-communication malicious-secure multi-server protocol for securely testing that a client-provided secret-shared vector is in $\mathcal{L}$.

We also prove the first nontrivial lower bounds showing limits on the sketch size for certain languages (e.g., vectors of Hamming-weight one) and proving the non-existence of arithmetic sketch schemes for others (e.g., the language of all vectors that contain a specific value).

Collaborative zero knowledge proofs.

At Usenix 2022 we presented a paper on "collaborative zero knowledge proofs" [OB22]. The paper, authored by Dan Boneh and Alex Ozdemir, studies how to generate a zero knowledge proof when the witness $w = (w_1, \ldots, w_n)$ is distributed among n provers $P_1, \ldots, P_n$, where prover P_i knows w_i. The provers want to generate a zero knowledge proof of knowledge of the witness w, without any prover revealing anything else to the other provers. This question comes up in many situations, especially when a weak device wants to outsource the generation of a ZK proof to a set of untrusted servers.

Our approach to building a collaborative proof is to apply an MPC protocol to the proof algorithm of a ZK system. The challenge is to construct a ZK proof system where the prover is MPC friendly. We show how to optimize a number of ZK proof systems for this goal, and we implement and experiment with the resulting collaborative proof systems. Our work gives an ordering of ZK proof systems based on their optimized MPC friendliness. In particular, we show that collaborative proofs are quite practical. Over a fast network, our collaborative proof systems are almost as fast as a standalone proof system. The reason is that, due to our optimizations, the provers spend most of their time doing local computations.

Other work.

Use or disclosure is subject to the restrictions on the cover of this document.

In recent years there has been strong interest in post-quantum cryptography, and in particular post-quantum signatures. NIST recently selected three post-quantum signature schemes to be standardized (Dilithium, Falcon, and Sphincs). All three produce signatures that are considerably longer than existing signature schemes used in practice, such as EdDSA and BLS. This makes them difficult to use in certificate chains and in many other applications. The question is how to build an efficient *short* post-quantum signature scheme. One promising approach is to use cryptographic group actions. Unfortunately, our work at Eurocrypt 2023 [BGZ23] provides a lower bound on the length of signatures built from group actions. The bound shows that despite its initial promise, this is not a viable direction for short signatures.

At Crypto 2022 [BK22] we showed how to construct a threshold signature scheme that provides *private* accountability. The scheme ensures that a signature reveals nothing to the public about the quorum that generated it, but provides full accountability to a party in possession of a secret tracing key. The construction makes heavy use of modern zero-knowledge SNARK techniques.

1.2.2 CMU

Piano: Extremely Simple PIR with Sublinear Server Computation.

We construct Piano, the first practical Private Information Retrieval (PIR) scheme with sublinear server computation. To the best of our knowledge, all prior implementations of PIR suffer from linear server computation per query. Piano is extremely simple, and concretely it's orders of magnitude more efficient than the state-of-the-art PIR scheme. Experimental results show that Piano incurs 99ms response time for a 100GB dataset, while the prior best scheme requires 11s — the performance number for the prior scheme is extrapolated since the prior implementation cannot even support such large databases. Our work represents a paradigm shift for practical PIR, since we can now scale up PIR to large datasets.

Since, we published our paper on Eprint [ZPSZ23], we've already received inquiries from big companies who are interested in deploying our scheme in real-world applications.

Our implementation is open sourced at `https://github.com/pianopir/Piano-PIR`

Efficient distributed zero-knowledge proof for the shuffle model.

Companies like Google and Meta use the shuffle model to enable large-scale privacy-preserving federated learning. In the shuffle model, a trusted shuffler (either enabled by cryptography or trusted hardware) permutes the inputs from all clients, such that each client's anonymity is protected. Each client may contribute one or more inputs. A line of previous works has shown that the shuffle model can significantly amplify privacy in federated learning. One important question is how to defend against malicious clients who might submit rogue data. Ideally, we would like to have all the clients prove that the inputs they have submitted satisfy certain sanity checks. In ongoing work, we are designing and implementing an efficient distributed zero-knowledge proof scheme for the shuffle model.

1.3 Task 6. Collaborations and Management

SRI and Tel Aviv University (TAU) are still finalizing the new contract that would continue the collaboration thread with Rafael Pass that delivered interesting results in the applications of SPARKS [EFKP20] into ZKgen, particularly by enriching the LPZK verified implementation with parallelism.

Collaborations within EMPHASIZE.

- Following the collaboration between SRI and Cornell on parallelized proofs, SRI plans to resume that work collaboratively with TAU as soon as TAU's contract is signed. The formal method approach for ZKgen necessitates solid theoretical foundations for parallelization, which the work from Cornell/TAU provides (e.g., PRAM model). The work at Cornell/TAU will further allow ZKgen to leverage parallelization beyond the split of IR circuits into independent pieces of computation.

Use or disclosure is subject to the restrictions on the cover of this document.

We have also started collaborating on Cornell/TAU continuous VDF construction to leverage parallelism; the ambition is to turn Cornell/TAU python prototype into an integrated component of ZKgen.

- SRI is still collaborating with CMU on the integration of Nanogram into ZKgen.
- SRI has resumed its collaboration with Stanford on PCS proof systems.

Collaborations with other SIEVE performers.

In this quarter, we continued the collaboration with the WIZKIT team that we started in Phase 2. The main effort was a paper submission for CCS'2023 and its revision, describing the formally verified LPZK work, as mentioned in Section 1.1.3. The paper has been assessed as requiring minor revision. The next collaboration phase with WIZKIT still concerns the objective of formally verifying a concrete vector linear oblivious evaluation (VOLE) protocol to be used in the correlated randomness generation process required by LPZK.

We will also continue collaborating with all other SIEVE performers on the IR.

2 Publications

- Mingxun Zhou, Andrew Park, Elaine Shi and Wenting Zheng. *Accountable Threshold Signatures with Proactive Refresh.* Eprint [ZPSZ23].
- Alex Ozdemir, Evan Laufer, and Dan Boneh. *zkpi: Proving lean theorems in zero-knowledge.* Submitted to IEEE S&P 2023. [OLB23]
- Alex Ozdemir, Gereon Kremer, Cesare Tinelli, and Clark W. Barrett. *Satisfiability modulo finite fields. IACR Cryptol. ePrint Arch.*, page 91, 2023. [OKTB23]

3 Planned Activities and Milestones

The plans for developing ZKgen, in the next quarter and beyond, include:

- Continue the collaboration with CMU on the integration of NanoGRAM, leveraging the RAM plugin in the SIEVE IR0+ format.
- In the formally verified approach, collaborate with WIZKIT on a formally verified approach to the generation of correlated randomness.
- Explore new extraction pathways from EasyCrypt, seeking greater performance, for instance targetting C or Rust directly instead of OCaml.
- Consider migrating the PCS-based implementation's sieve IR proccessing frontend from zki_interface to wiztoolkit.
- Deepen the collaboration with Stanford to implement more efficient PCS to plug into Plonk, such as *Efficient polynomial commitment schemes for multiple points and polynomials*, by Boneh, Drake, Fisch and Gabizon [BDFG20], as well as PCS optimizations such as *Plookup* [GW20].

We still also plan to optimize the circuits given to ZK-gen as inputs, by using a component of SRI's SMT-solver Yices that is designed to do that for bitvector circuits (arithmetic modulo 2^n) and could be generalized to IR0 or simple IR1 circuits.

Use or disclosure is subject to the restrictions on the cover of this document.

Appendix A: References

[BBC+23] Dan Boneh, Elette Boyle, Henry Corrigan-Gibbs, Niv Gilboa, and Yuval Ishai. arithmetic sketching. 2023. in submission.

[BDFG20] Dan Boneh, Justin Drake, Ben Fisch, and Ariel Gabizon. Efficient polynomial commitment schemes for multiple points and polynomials. Cryptology ePrint Archive, Report 2020/081, 2020. https://eprint.iacr.org/2020/081.

[BGZ23] Dan Boneh, Jiaxin Guan, and Mark Zhandry. A lower bound on the length of signatures based on group actions and generic isogenies. In *Eurocrypt '23*, 2023.

[BK22] Dan Boneh and Chelsea Komlo. Threshold signatures with private accountability. In *Crypto '22*, volume 13510 of *Lecture Notes in Computer Science*, pages 551–581. Springer, 2022.

[CBBZ23] Binyi Chen, Benedikt Bünz, Dan Boneh, and Zhenfei Zhang. Hyperplonk: Plonk with linear-time prover and high-degree custom gates, 2023. full version `eprint.iacr.org/2022/1355`.

[DB22] Trisah Datta and Dan Boneh. Using zk proofs to fight disinformation. Medium , 2022.

[DILO22] Samuel Dittmer, Yuval Ishai, Steve Lu, and Rafail Ostrovsky. Improving line-point zero knowledge: Two multiplications for the price of one. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022*, pages 829–841. ACM, 2022.

[EFKP20] Naomi Ephraim, Cody Freitag, Ilan Komargodski, and Rafael Pass. Sparks: Succinct parallelizable arguments of knowledge. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part I*, volume 12105 of *Lecture Notes in Computer Science*, pages 707–737. Springer, 2020.

[GMO16] Irene Giacomelli, Jesper Madsen, and Claudio Orlandi. Zkboo: Faster zero-knowledge for boolean circuits. Cryptology ePrint Archive, Paper 2016/163, 2016. https://eprint.iacr.org/2016/163.

[Gro16] Jens Groth. On the size of pairing-based non-interactive arguments. Cryptology ePrint Archive, Paper 2016/260, 2016. https://eprint.iacr.org/2016/260.

[GW20] Ariel Gabizon and Zachary J. Williamson. plookup: A simplified polynomial protocol for lookup tables. Cryptology ePrint Archive, Report 2020/315, 2020. https://ia.cr/2020/315.

[LSTW21] Jonathan Lee, Srinath Setty, Justin Thaler, and Riad Wahby. Linear-time and post-quantum zero-knowledge snarks for r1cs. Cryptology ePrint Archive, Paper 2021/030, 2021. https://eprint.iacr.org/2021/030.

[OB22] Alex Ozdemir and Dan Boneh. Experimenting with collaborative zk-snarks: Zero-knowledge proofs for distributed secrets. In Kevin R. B. Butler and Kurt Thomas, editors, *31st USENIX Security Symposium*, pages 4291–4308. USENIX Association, 2022.

[OKTB23] Alex Ozdemir, Gereon Kremer, Cesare Tinelli, and Clark W. Barrett. Satisfiability modulo finite fields. *IACR Cryptol. ePrint Arch.*, page 91, 2023.

Use or disclosure is subject to the restrictions on the cover of this document.

[OLB23] Alex Ozdemir, Evan Laufer, and Dan Boneh. zkpi: Proving lean theorems in zero-knowledge. in submission, 2023.

[YSWW21] Kang Yang, Pratik Sarkar, Chenkai Weng, and Xiao Wang. Quicksilver: Efficient and affordable zero-knowledge proofs for circuits and polynomials over any field. In Yongdae Kim, Jong Kim, Giovanni Vigna, and Elaine Shi, editors, *CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, Republic of Korea, November 15 - 19, 2021*, pages 2986–3001. ACM, 2021.

[ZPSZ23] Mingxun Zhou, Andrew Park, Elaine Shi, and Wenting Zheng. Piano: Extremely simple, single-server pir with sublinear server computation. Cryptology ePrint Archive, Paper 2023/452, 2023. https://eprint.iacr.org/2023/452.

[ZXZS19] Jiaheng Zhang, Tiancheng Xie, Yupeng Zhang, and Dawn Song. Transparent polynomial delegation and its applications to zero knowledge proof. Cryptology ePrint Archive, Paper 2019/1482, 2019. https://eprint.iacr.org/2019/1482.

Use or disclosure is subject to the restrictions on the cover of this document.