Research Statement

Yibin Yang

In an era where information is the new currency, data sharing is indispensable for both operational efficiency and groundbreaking innovation. For example, sharing patient records across multiple hospitals not only improves individual care but also supports large-scale health analytics; electric utility companies rely on shared data to optimize power flow, cutting costs and enhancing grid reliability; organizations can collaboratively train ML models to improve service quality across the board; and in research, access to comprehensive datasets, such as those related to rare diseases, is crucial for developing new treatments. However, despite these benefits, concerns related to intellectual property rights, strict privacy laws, security risks, and complex data custody often hinder this valuable sharing.

In principle, cryptography offers solutions to address most of these concerns. *Zero-Knowledge Proofs* (ZKP) allow one party, the prover, to prove *any* property about its private data to another party, the verifier, without disclosing any additional information. More generally, *Secure Multi-Party Computation* (MPC) enables multiple mutually untrusting parties to jointly compute *arbitrary* functions on their private inputs, revealing only the intended output. While these cryptographic techniques hold great theoretical promise, both ZKP and MPC face significant challenges in real-world adoption. These challenges stem from their complex algorithms, which require specialized, PhD-level expertise, and their often poor performance, making practical deployment difficult or even impossible.

My Vision for Efficient and Accessible ZKP and MPC

I specialize in **applied ZKP and MPC**, developing efficient and usable cryptographic systems. In a virtuous cycle, building systems provides me with insights into the obstacles and bottlenecks in ZKP and MPC, which I eliminate or mitigate through my algorithmic and theoretical research.

I am uniquely positioned at the **intersection of theory and systems**, with a strong ability to bridge the gap between these two realms. My expertise lies not only in developing rigorous novel cryptographic primitives with proper definitions and proofs, but also in translating these formal constructions into easy-to-use, efficient, secure implementations.

My **career goal and vision** is to make advanced ZKP and MPC (and cryptography in general) **easily, efficiently, and securely accessible** for everyone, including those with "zero knowledge" in the field. My work towards achieving this falls under a core principle and two broad approaches:

Co-designing algorithms and systems. This is an overarching principle I apply in my work. I believe it is essential that my cryptographic algorithm research and prototype system building develop in unison and inform each other's direction and promising techniques.

Indeed, building a reasonably efficient ZKP or MPC system critically relies on the new fundamental primitives I developed (e.g., efficient ZKP/MPC branching and read-write memory). Using prior primitives would have resulted in a performance degradation of several orders of magnitude. Conversely, I developed these primitives specifically to address pain points identified during system development.

ZKP and MPC toolchains for everything and everyone. State-of-the-art ZKP and MPC protocols typically accept programs encoded as *circuits*, because known protocols for other encoding representations are significantly less efficient. However, for good reasons, real-world programs are specified using high-level programming languages (e.g., `C`/`C++`/Assembly). I will refer to such programs as *random-access machine* (RAM) programs, highlighting their key strength (and their high ZKP/MPC cost). While any RAM program can be compiled into a circuit, doing so leads to a very large circuit.

Optimizing the circuit for a specific task is occasionally possible, but even then, it requires extensive expertise in cryptography and related fields, which hinders the deployment of ZKP and MPC. Instead, I believe that ZKP and MPC should directly and efficiently execute RAM programs, with compilers that translate high-level languages into a suitable ZK/MPC intermediate representation (IR). Then, engineers without full expertise in cryptography can develop a customized privacy-preserving application simply by writing a *regular* program in a language *they are already familiar with*.

Achieving this is challenging. First, it requires the development of fundamentally new techniques since circuits are inefficient representations for RAM programs: they omit crucial elements, such as conditional branching (i.e., `if` statements) and memory access operations (i.e., `load`/`store` statements). Second, building secure, efficient, usable, and maintainable toolchains using these new techniques is a substantial engineering undertaking.

ZKP and MPC libraries for special tasks for everyone. Some basic functions, e.g., set intersection and database operations, can be (much) more efficiently computed using specially designed protocols. My second approach is to identify these crucial and frequently needed components, design optimized protocols, and provide standalone implementations that can be easily integrated into general toolchains.

Achieving this is once again a challenge due to the wide variety of applications and use scenarios, such as the popular cloud computing and the evolving field of blockchain technology. Each area may require *different* building blocks. Crucially, these building blocks should be as general as possible to offer a certain level of *flexibility* for each domain, such as allowing users to program application-specific conditions and switch between classical and post-quantum instantiations.

Guided by these ideas, **my research program is dedicated to turning my vision into reality.** The rest of this statement is divided into two parts: §1 reviews my completed research projects that have advanced state-of-the-art towards this vision, and §2 includes my future research plans.

ZKP and MPC performance in context. I briefly review some key metrics to illustrate the kinds of ZKP and MPC tasks that could be handled *on commodity hardware* (such as a laptop) prior to my work, the capabilities I achieved, and where I envision progress over the next few years.

I developed the first efficient end-to-end ZKP toolchain capable of proving statements expressed as `ANSI C` programs at speeds >10KHz. It executed off-the-shelf Linux programs such as `gzip` — achieving proof of a real-world `gzip` vulnerability (reported in CVE) in ≈ 5 seconds. Prior work executed at speeds <10Hz, leaving the execution of "real `C` programs" out of reach. I developed the first efficient end-to-end 2PC toolchain that executes computational tasks written in Assembly language, supporting full memory access and general control flow, at $\approx$ 1KHz — a 1000× improvement over prior work.

I plan to integrate my recent and future work into these toolchains. I anticipate achieving an additional 2–4 orders of magnitude improvement on them, reaching speeds in the (tens of) MHz range. This is a qualitative leap, paving the way for widespread tool adoption and unlocking transformative large-scale privacy and data-sharing applications, e.g., involving patients in managing access to their hospital-held DNA records, proving in ZK the fairness of larger ML model decisions, and more.

1 Overview of Completed Research

1.1 ZKP and MPC Toolchains for Everything and Everyone

Challenges in ZKP and MPC for real-world RAM programs. A RAM program repeatedly executes instructions from a fixed instruction set and allows random access to a large read-write memory. Casting this to ZKP and MPC, two significant overheads arise, which I address in my work:

- **Executed Instruction:** A cleartext RAM program executes only one instruction per step. Within ZKP and MPC, it is necessary to conceal which instruction is being executed. The naïve solution is to execute every machine instruction at each step, leading to substantial overhead.
- **Memory Access Pattern:** Similarly, a cleartext RAM program accesses a single memory location per step. Within ZKP and MPC, the accessed address (and value) must be hidden. Current approach involves using expensive *Oblivious RAM* (or just a full linear scan per access).

ZKP toolchains for `ANSI C`. I developed practically efficient ZKP systems capable of accepting *any* `ANSI C` program as input to encode the proved statement [HYDK21, YHKD22]. These systems are the *first* that directly execute *off-the-shelf* standard Linux programs in ZK (e.g., prove published CVE vulnerabilities) and are still the fastest in end-to-end time. I addressed the above challenges as follows:

In [HYDK21], I carefully selected a ZKP-friendly instruction set, ensuring that the total circuit size remains reasonably small for per-step execution. I designed efficient ZK read-write memory by leveraging the standard Waksman permutation networks and the locality of memory access, reducing the per-step cost to logarithmic. Crucially, I implemented a full toolchain that starts with an `ANSI C`

program, compiles it into a ZKP-friendly assembly, and executes it within a ZKP virtual machine. The resulting system can execute over 10K instructions per second in a LAN setting on a laptop. A ZK bug bounty proof of concept, it successfully executed off-the-shelf buggy versions of Linux programs `sed` and `gzip`, proving in ZK that each program contains a CVE-reported bug in approximately 30 and 5 seconds, respectively. In the follow-up work [YHKD22], I further enhanced this system by exploring parallelism, increasing the execution speed to over 300K instructions per second.

This line of work demonstrates the **practical feasibility** of my first approach for ZKP. Next, I discuss several of my more recent advanced techniques for handling both instructions and memory in ZK which are (nearly) asymptotically optimal and concretely efficient.

Innovative methods for ZK branching. Executing each step in a RAM program involves executing one instruction from the full instruction set. In ZKP, a statement corresponding to instruction execution is a disjunction of B different instructions, where a single *active* clause is satisfied.

In [YHH+23], I introduced Robin, a ZKP protocol for a single disjunction, whose communication is proportional to its *largest* clause. It improved the then-state of the art by $\approx 3\times$. More importantly, RAM programs repeatedly execute *identical* disjunctions representing the instruction set. In my ZK protocol Batchman (also in [YHH+23]), I achieved asymptotic (and concretely efficient!) computation improvement for batched disjunction settings. Specifically, for a batch size R, Batchman incurs computational costs proportional to $R + B$ rather than the previous $R \cdot B$. These protocols are highly generic and can be instantiated in any standard *commit-and-prove* ZK system, including the cutting-edge VOLE-based ZK with the shortest end-to-end time. Instantiated with the VOLE-based ZK, Batchman achieved a concrete improvement of over $70\times$ compared to previous approaches for handling batched disjunctions or executed instructions.

[YHH+23] is the *first* to show the potential for computation savings in batched disjunctive statements and was honored with the **Distinguished Paper Award** at ACM CCS 2023.

In subsequent work [HHK+24], I further optimized the non-batched disjunction setting targeting VOLE-based ZK. I proposed a protocol called LogRobin++, which outperforms Robin by more than $3\times$. Notably, LogRobin++ incurs only an $\mathcal{O}(\log B)$ additive term in communication, which is nearly *optimal* since encoding the index of the active clause requires $\log B$ bits even without ZK.

State-of-the-art ZK read-write memory. My ZKP toolchains [HYDK21, YHKD22] motivated a line of work on efficient ZK read-write memory, showing it can be implemented with amortized constant overhead per access, surpassing the logarithmic overhead presented in [HYDK21].

Among them, my work [YH24] is the **state of the art**. In [YH24], I showed that a lifetime of ZK read-write memory can be *information-theoretically* reduced to two proofs of a permutation. This construction is exceptionally lean: each access only requires amortized 10 input/multiplication gates.

Concretely, the ZK memory introduced in [YH24] outperformed the one in [HYDK21] by a factor of 12-160×, depending on the network settings. Furthermore, the construction in [YH24] establishes a connection between ZK memory and a long line of work on *offline memory checkers*. This link suggests that further optimization of ZK memory may be inherently challenging.

Integrating ZK branching and ZK memory for a tight ZK CPU. My work on batched ZK branching [YHH+23] and ZK memory [YH24] can be seamlessly integrated to provide a more efficient mechanism for executing RAM programs inside ZKP by repeatedly emulating CPU steps (ZK CPU). Moreover, I presented a novel approach in [YHH+24] to this integration, resulting in a *tight* ZK CPU:

- **Tight Cost:** In the Batchman protocol [YHH+23], the cost of each step is proportional to the *largest* CPU instruction. In [YHH+24], this cost is proportional to the *executed* instruction. This allows for efficient handling of CPUs with vast instruction-size differences. Such CPUs naturally arise based on the program's control-flow graph [YHH+24].
- **Tight Leakage:** Existing ZKPs for RAM programs reveal the number of executed instructions to the verifier. [YHH+24] shows that it is possible to only reveal the total number of executed multiplication gates, enhancing privacy.

[YHH+24] is **unique** in achieving the above tightness. Essentially, it implies that the ZKP of executing a RAM program incurs only a constant factor penalty compared to plaintext execution — a major step toward my first approach as applied to ZKP.

Efficient constant-round 2PC for assembly programs. *Secure Two-Party Computation* (2PC) is a special case of MPC. In my work [YPHK23], I developed the *first* constant-round 2PC protocol capable of executing *any* assembly RAM program, accompanied by *practically efficient* implementations.

Unlike ZKP for RAM programs, where only one party's privacy needs protection, 2PC scenarios require safeguarding the privacy of both participants, presenting additional challenges.

Garbled Circuits (GCs) are fundamental building blocks for MPC. The two essential techniques for efficiently handling disjunctions and memory within GCs — *stacked garbling* and *garbled RAM* — are *incompatible* with each other. In [YPHK23], I designed a novel scheme combining their advantages.

The resulting protocol achieves over 1000× performance improvements compared to the prior state of the art on WAN-like high-latency networks. Additionally, it provides *the first efficient implementation* of garbled RAM. [YPHK23] demonstrates the **practical feasibility** of my first approach for MPC.

As a proof of concept, I implemented efficient 2PC versions of Dijkstra and KMP. Development was as straightforward as writing regular Assembly programs, without handling any cryptographic details.

1.2 ZKP and MPC Libraries for Special Tasks for Everyone

It is important to note that my works in ZK branching [YHH+23, HHK+24], ZK memory [YH24], and garbled RAM [YPHK23] are each designed and packaged as a *standalone open-sourced* library. For instance, my ZK memory [YH24] has already been adopted by other ZK-ML researchers. This section provides an overview of other MPC libraries that I have developed.

Versatile post-quantum oblivious PRFs. The *Oblivious Pseudorandom Function* (OPRF) is a highly significant and widely deployed special-purpose 2PC primitive. Notable applications include *key management systems* and *password-based key exchange*. However, developing a *practically efficient* post-quantum OPRFs has remained a *longstanding* open problem.

During my summer research internship at AWS in 2024, I devised an **efficient post-quantum OPRF** [YBH+24]. This versatile OPRF library offers efficient switching between batched and non-batched evaluations, verifiable and non-verifiable modes, and classical and post-quantum security.

Programmable Ethereum money in payment channels. Scalability remains the primary bottleneck in blockchain ledgers. *Payment channels* process transactions on a layer-2 network, achieving scalable payments. However, payment channels do not support programmability (i.e., *smart contracts*), forgoing one of the most attractive features of modern blockchains.

During my summer internship at Visa Research in 2021, I designed a protocol to enable payment channels with **arbitrary layer-2 programmability on Ethereum** [KLM+24]. Unlike previous approaches, which are limited to a predetermined payment logic, my work [KLM+24] allows users to specify *arbitrary* payment logic after the programmable payment channels are established. Based on this, I developed scalable layer-2 auction protocols [MLK+23].

1.3 Exploring the Barriers in MPC

Advancing the frontier of MPC occasionally leads to setbacks. While this can be disappointing, it inspired me to develop impossibility proofs and lower bounds.

Malicious 2PC via arithmetic garbling. Recent progress in garbled *arithmetic* circuits enabled constant rate (i.e., constant factor overhead) for the *bounded integer computation* model.

During my summer visit to Bar-Ilan University in 2023, I explored the feasibility of attaining malicious-secure 2PC using this technique. I showed that in this model, malicious security can be achieved only against a single party, and further, one bit of information must be leaked [HY24]. I complemented this lower bound by proposing the **first** constant-rate, constant-round 2PC protocol, attaining this level of optimal security.

Fair MPC in the unreactive world. When more than half of the parties are corrupted, Cleve's 1986 seminal lower bound establishes that achieving fair MPC in the plain model is impossible: the adversary can always learn the MPC output while hiding it from honest parties.

During my summer internship at Visa Research in 2022, I investigated whether unreactive *trusted third parties*, connected to only subsets of parties, could circumvent this lower bound. I provided a **comprehensive characterization** in [RY23], which relies on *novel* insights from schoolbook Cleve's.

2 Outlook of Future Research

Next-generation ZKP toolchains for everything and everyone. My work on ZK branching [YHH+23] and ZK memory [YH24] teases the possibility of development of a new generation of

ZKP toolchains with dramatically larger scale, capable of accepting most (or even all!) off-the-shelf real-world programs. In the initial phase, my work [YHH+24] has successfully reduced the overhead of executing these programs within ZKP to only a *constant factor*. I am enthusiastic about continuing to advance these second-generation toolchains to scale up and extend [HYDK21] to support far larger programs, and to provide robust security and privacy solutions for real-world applications.

This area presents numerous *intriguing* and *practically valuable* research opportunities. For example, my ZK CPU [YHH+24] (especially its *non-black-box* use of my ZK memory) can be further improved, achieving a (smaller) constant factor overhead over a plaintext execution. Additionally, it operates over fields, whereas most real-world computations are defined over integer rings. My ongoing research has already removed several obstacles, but further investigation is needed, and I am excited to proceed.

Advanced 2PC/MPC branching and memory. In contrast to ZKP, developing efficient 2PC/MPC solutions for real-world programs still requires new fundamental techniques. Just as [HYDK21] has inspired my subsequent research, [YPHK23] highlights the necessity of exploring 2PC/MPC branching and memory. The exploration is inherently more challenging than ZK. I am passionate about creating *practical* 2PC/MPC toolchains that facilitate secure and privacy-preserving collaboration and are fast and easy-to-use for all engineers. For example, my ongoing research explores new approaches to achieving more efficient garbled RAM and integration of compiler technologies for better MPC.

ZKP and MPC libraries in cloud computing and the LLM era. Informed by my internship at AWS, I am interested in developing and implementing ZKP and MPC libraries tailored for cloud computing applications. Traditional cloud computing has predominantly emphasized storage solutions; however, the current trend is shifting towards enhancing computational capabilities. It is increasingly important and beneficial for cloud servers to provide not only efficiency but also *robustness* and/or *privacy* in these computational processes, areas where ZKP and MPC can be naturally applied. These computations can range from lightweight tasks, such as key management, to more intensive operations involving multiple cloud servers collaboratively computing on encrypted data.

This trend is further amplified by the emergence of *Large Language Models* (LLMs). LLMs, such as ChatGPT, are cloud-based services with small communication but tremendous computation. As LLMs are increasingly deployed in critical applications, it becomes imperative to safeguard the privacy of user queries. ZKP and MPC present potential solutions for enhancing privacy in (some of) these contexts. However, a significant efficiency gap remains, which must be bridged by developing new cryptographic notions, assumptions, primitives, and other innovations. I am passionate about contributing my expertise and effort to this research direction.

A tool for broadening participation. I would like to highlight a perspective that drives my goal of achieving "ZKP and MPC for everything and everyone." I believe that developing effective ZKP and MPC toolchains for high-level programming languages can make advanced cryptography broadly accessible to non-cryptographer engineers of all backgrounds. This accessibility would promote wider adoption, solve or mitigate privacy concerns in various applications, and ultimately enhance inclusivity. For example, MPC can be used to improve student disability accommodations through more comprehensive, secure data collection and analysis. My research aims to accelerate this development.

Interdisciplinary collaboration. My research and vision benefit from interdisciplinary collaboration across various domains of computer science and beyond. While I have already built complete `ANSI C` and Assembly toolchains for ZKP and MPC, respectively, a natural next step for performance improvements is to co-design cryptographic, *compiler* and *programming language* components. Immediate research topics include designing (1) tailored programming languages or annotations that allow the programmer to guide ZKP or MPC optimizations, (2) improved intermediate representations, and (3) the generation of basic blocks and control-flow graphs optimized for ZKP or MPC backends. To ensure these toolchains are highly standardized and user-friendly, integration with fields such as *systems*, *software engineering*, and *human-computer interaction* is essential.

Conversely, ZKP and MPC can be applied to problems arising in other fields, wherever trust needs to be established or sensitive data needs to be shared. Obvious examples include the medical field, machine learning, sensitive statistical studies, etc.

Naturally, I maintain a broad interest in *cryptography* beyond ZKP and MPC, and *security* overall.

I am enthusiastic about collaborating with faculty, experts, researchers, undergraduate and graduate students, and influencing industries with diverse backgrounds to advance ZKP and MPC toolchains and libraries, making them accessible and effective for everyone, and applying cryptography in other fields.

References

[HHK+24] Carmit Hazay, David Heath, Vladimir Kolesnikov, Muthuramakrishnan Venkitasubramaniam, and Yibin Yang. LogRobin++: Optimizing proofs of disjunctive statements in VOLE-based ZK. In *ASIACRYPT 2024*, LNCS. Springer, 2024.

[HY24] Carmit Hazay and Yibin Yang. Toward malicious constant-rate 2PC via arithmetic garbling. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part V*, volume 14655 of *LNCS*, pages 401–431. Springer, Cham, May 2024.

[HYDK21] David Heath, Yibin Yang, David Devecsery, and Vladimir Kolesnikov. Zero knowledge for everything and everyone: Fast ZK processor with cached ORAM for ANSI C programs. In *2021 IEEE Symposium on Security and Privacy*, pages 1538–1556. IEEE Computer Society Press, May 2021.

[KLM+24] Ranjit Kumaresan, Duc Viet Le, Mohsen Minaei, Srinivasan Raghuraman, Yibin Yang, and Mahdi Zamani. Programmable payment channels. In Christina Pöpper and Lejla Batina, editors, *ACNS 24International Conference on Applied Cryptography and Network Security, Part III*, volume 14585 of *LNCS*, pages 51–73. Springer, Cham, March 2024.

[MLK+23] Mohsen Minaei, Duc V. Le, Ranjit Kumaresan, Andrew Beams, Pedro Moreno-Sanchez, Yibin Yang, Srinivasan Raghuraman, Panagiotis Chatzigiannis, and Mahdi Zamani. Scalable off-chain auctions. Cryptology ePrint Archive, Report 2023/1454, 2023.

[RY23] Srinivasan Raghuraman and Yibin Yang. Just how fair is an unreactive world? In Jian Guo and Ron Steinfeld, editors, *ASIACRYPT 2023, Part VI*, volume 14443 of *LNCS*, pages 420–450. Springer, Singapore, December 2023.

[YBH+24] Yibin Yang, Fabrice Benhamouda, Shai Halevi, Hugo Krawczyk, and Tal Rabin. Gold OPRFs: Post-quantum oblivious power residue PRF, 2024. Under submission.

[YH24] Yibin Yang and David Heath. Two shuffles make a RAM: Improved constant overhead zero knowledge RAM. In Davide Balzarotti and Wenyuan Xu, editors, *USENIX Security 2024*. USENIX Association, August 2024.

[YHH+23] Yibin Yang, David Heath, Carmit Hazay, Vladimir Kolesnikov, and Muthuramakrishnan Venkitasubramaniam. Batchman and robin: Batched and non-batched branching for interactive ZK. In Weizhi Meng, Christian Damsgaard Jensen, Cas Cremers, and Engin Kirda, editors, *ACM CCS 2023*, pages 1452–1466. ACM Press, November 2023.

[YHH+24] Yibin Yang, David Heath, Carmit Hazay, Vladimir Kolesnikov, and Muthuramakrishnan Venkitasubramaniam. Tight ZK CPU: Batched ZK branching with cost proportional to evaluated instruction. In *ACM CCS 2024*. ACM Press, 2024.

[YHKD22] Yibin Yang, David Heath, Vladimir Kolesnikov, and David Devecsery. EZEE: Epoch parallel zero knowledge for ANSI C. In *2022 IEEE European Symposium on Security and Privacy*, pages 109–123. IEEE Computer Society Press, June 2022.

[YPHK23] Yibin Yang, Stanislav Peceny, David Heath, and Vladimir Kolesnikov. Towards generic MPC compilers via variable instruction set architectures (VISAs). In Weizhi Meng, Christian Damsgaard Jensen, Cas Cremers, and Engin Kirda, editors, *ACM CCS 2023*, pages 2516–2530. ACM Press, November 2023.