SaUCy: Super Amazing Universal Composability

Kevin Liao

University of Illinois Urbana-Champaign

Blockchains often comprise an array of advanced primitives from distributed computing and cryptography, including consensus protocols [1], zero-knowledge proofs [2], and multiparty computation [3]. However, securely composing these primitives is difficult—vulnerabilities often arise from misunderstandings and mismatches as components are integrated [4]. Because blockchains handle billions of dollars in financial assets and provide security critical services (e.g., privacy) to their users, we should be assured with mathematical certainty that they are free of such vulnerabilities.

In cryptography, this problem has been addressed in the theory of universal composability (UC), which is considered the gold standard for demonstrating that a protocol "does its job securely" [5]. In particular, a UC-secure protocol maintains its security properties in any context, and so we can analyze complex protocols in a modular way from simpler building blocks. Even so, designing UC protocols and proving them secure with pen and paper remains highly error-prone, let alone preserving on-paper guarantees in concrete implementations. One solution for addressing these challenges is through formal verification, where the gold standard is mechanized proof (every step has been fully computer-checked). Thus, to meet the demanding security requirements of blockchains, this project will develop new techniques for combining UC security and mechanized proof, with an eye towards building practical, provably secure protocol implementations.

Our Approach. The aim of this project is to develop a framework called SaUCy (short for "Super Amazing Universal Composability") for building secure-by-construction systems in modular fashion. Inspired by projects such as IronFleet [6] and Everest [7], SaUCy will provide a toolchain for mechanically verifying the security of abstract components, and for refining them into performant, correct low-level code. To further guarantee that these components compose securely in larger systems, SaUCy will adopt the theory of universal composability as its central organizing principle, thus categorically eliminating "bad interactions" arising between otherwise secure components. Altogether, SaUCy will present a new modus operandi for building secure systems—ones that are more ambitious and have greater assurance than before—without compromising on their performance or development cost.

We will demonstrate the SaUCy methodology through building provably secure blockchains, many of which suffer from increasingly unwieldy formalisms. Of particular interest are blockchains that require secure composition "in the large" to achieve the desired security and liveness properties. The Dfinity blockchain would be an ideal case study—the distributed key generation mechanism alone combines verifiable secret sharing, zero knowledge proofs, polynomial commitments, and blockchain consensus [8]. Using the SaUCy methodology, we will be able to give a formal specification of the Dfinity blockchain, produce a mechanized proof of UC security, and refine the abstract specification into a performant, correct low-level implementation—all without compromises. Of course, the benefits of UC-security go beyond the Dfinity blockchain. The whole idea of UC is to ensure that protocols compose securely with arbitrary other protocols, and so we can also ensure that external systems interacting with the Internet computer can do so in a provably secure way.

Roadmap. Building out the SaUCy framework will require technical advances at the boundary of formal methods, cryptography, and distributed computing along three thrusts. To summarize, the first thrust focuses on building a concrete, executable implementation of the UC framework as a stepping stone to the second thrust, which includes formalizing and mechanizing UC security proofs. Having laid the foundations for SaUCy, the third thrust explores integrations with existing tools and techniques for building verified implementations.

Thrust 1: ***Building a concrete, executable implementation of UC.***

The goal of this thrust is to build a concrete, executable implementation of the UC framework. This is important for two reasons: First, an executable implementation is a first step towards formal, mechanized proofs. Second, being able to "run" proof constructions is useful for sanity checking and for finding counterexamples for where they may break down.

In preliminary work, we have designed a specialized process calculus called the Interactive Lambda Calculus (ILC), which we use to implement the UC execution model and to formally specify UC proof artifacts. Without getting into too many details, a UC security proof involves two steps. The first is a constructive step, which requires specifying the real world protocol to be analyzed, an ideal world protocol serving as the specification, and a simulator that translates attacks on the real world protocol to the ideal world protocol. ILC addresses the problem of how we can write these down in a formal way. The second step is a relational analysis between executions of the real and ideal world protocols, which we tackle in Thrust 2.

Why do we need a new programming language for UC in the first place? The main design idea of ILC is to faithfully abstract interactive Turing machines (ITMs), the computational model underlying UC. The significance of ITMs is that they have a clear computational interpretation, so it is straightforward to relate execution traces to a probabilistic polynomial time computation, as is necessary for cryptographic reduction proofs. The presence of non-probabilistic nondeterminism in alternative concurrency models (such as those based on the standard π-calculus [9]) would frustrate such reduction proofs. ITMs sidestep this issue by having a deterministic (modulo random coin tosses), "single-threaded" execution semantics. That is, processes pass control from one to another each time a message is sent so that *exactly one process is active at any given time*, and, moreover, *the order of activations is fully determined*.

In turn, ILC adapts ITMs to a subset of the π-calculus through an affine (certain resources are non-duplicable) typing discipline. In other words, *well-typed ILC programs are expressible as ITMs*. To maintain that only one process is active (can write) at any given time, processes implicitly pass around a unique, affine "write token" by virtue of where they perform read and write effects: When process A writes to process B, process A spends the write token and process B acquires it. Moreover, to maintain that the order of activations is fully determined, the read endpoints of channels are affine resources, and so each write operation corresponds to a single, unique read operation. Taken together, these type-level invariants give ILC its central metatheoretic property of *confluence*.

Confluence implies that the only nondeterminism in an ILC program is due to random coin tosses taken by processes, which have a well-defined distribution. Additionally, any apparent concurrency hazards, such as adversarial scheduling of messages in an asynchronous network, are due to an explicit adversary process rather than uncertainty built into the model itself. This eliminates non-probabilistic nondeterminism, and so ILC programs are amenable to the reasoning patterns necessary for establishing computational security guarantees.

Using ILC, we have implemented a prototype UC framework (similar to the Simplified UC framework [10]) and have ported over a sampling of theory from the UC literature. The full details of ILC and our prototype UC framework can be found in the attached paper (joint work with Matthew Hammer and Andrew Miller) [11].

Thrust 2: ***Formalizing and mechanizing UC security proofs.***

Thrust 1 tackles how to formalize the constructive step of a UC proof, so the goal of this thrust is to develop tools and techniques for formalizing and mechanizing the relational analysis step. Roughly, the relational analysis step consists of showing that executions of the real world protocol (the protocol under analysis) and the ideal world protocol (which runs a trusted third party called an *ideal functionality* and is thus trivially secure) are indistinguishable. In more detail, we must establish the difficulty of distinguishing between the two worlds by reduction from some computationally hard problem. Fortunately, ILC's strong confluence property enables reasoning about such reductions.

This indistinguishability analysis is essentially a comparison between executions of two programs, an idea that has been explored extensively in the area of relational program verification [12]. Relational verification is the central technique used in the EasyCrypt proof assistant, which supports constructing game-based security proofs (no compositionality guarantees) [13]. A recent result by Almeida et al. used EasyCrypt to mechanize a simulation proof for Yao's garbled circuit protocol [14], which suggests that relational verification could be extended to UC-style simulation proofs as well. We plan to embed ILC and our ILC implementation of UC in EasyCrypt (or even a more general-purpose verification tool such as Coq [15] or F* [16]) to use its tooling.

The challenges of formalizing and mechanizing UC can be tackled from the cryptographic perspective as well. Already, there is a growing body of work in making UC models simpler, more flexible, and more

usable [5, 17, 18, 19, 20, 21, 22, 10, 23, 24, 25, 26, 27]. Adding to these, another important goal of this thrust is to develop UC models that are more amenable to automated reasoning. A promising line of research is to investigate computational soundness [28] in the context of UC. The idea of computational soundness is to analyze protocols from a fairly abstract view (the symbolic model of cryptography), which can benefit from automated reasoning, and then show that security guarantees established in this formal world are preserved in the computational world.

Another promising line of research was investigated in a paper by Chase and Lysyanskaya [29], in which they introduce two security definitions for signatures of knowledge—a definition based on UC and a definition based on games—and ultimately prove them to be equivalent. Because formalizing and mechanizing game-based security proofs is relatively well-studied, such equivalence results may present a viable path towards mechanizing UC proofs. Thus, exploring what other primitives have equivalent UC and game-based security definitions will be important future work.

Thrust 3: ***Refining abstract protocols into performant, correct code.***

The current modus operandi for implementing high-performance cryptographic and distributed protocols requires an inordinate cost in both time and expertise. Thus, the goal of this final thrust is to leverage automated tools and techniques for refining abstract protocol specifications (written in ILC) into fast, low-level code that is provably correct, memory safe, and side-channel free.

We plan to implement an embedding of ILC in F*, a verification-oriented programming language that combines the automation of SMT with the expressive power of dependent types [16]. Because ILC programs will essentially be F* programs, we can use F*'s logic to verify memory safety, side-channel resistance, and functional correctness properties. Moreover, we can use various tools in the F* ecosystem to obtain platform-agnostic C implementations, or assembly implementations optimized for specific hardware targets. To obtain C implementations, ILC programs embedded in F* can be translated into C code using the KreMLin tool [30]. From there, the C code can be compiled using the CompCert compiler [31] (for minimizing the trusted computing base), or GCC/Clang (for maximizing performance). To obtain high-performance assembly implementations, expensive routines in ILC protocols can be refined into x64 assembly, which also has a verified embedding in F* [32].

How will developers use SaUCy-verified protocols to build secure systems? To the average developer, reasoning about asynchronous, distributed, and adversarial deployment environments is unnatural. To address this issue, we will build a module system around SaUCy that will simplify the task of composing distributed protocols and cryptographic primitives. The novel design idea is to include with each module a rich behavioral specification in the form of an ideal functionality, which serves as a self-contained specification of all desired security and liveness properties, along with the mechanized proof that the implemented protocol UC-realizes the ideal functionality. The payoff is that it should be much easier to reason about the ideal functionality, whereas the actual protocol (with multiple parties, inconsistent views, Bzyantine faults, and cryptography) is much messier. Thus, developers can read the ideal functionality to understand the protocol's guarantees and program against its interface. When the larger system is compiled, calls to the ideal functionality will be replaced by calls to the actual distributed implementation.

References.

[1] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. The honey badger of bft protocols. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 31–42. ACM, 2016.

[2] Ahmed Kosba, Andrew Miller, Elaine Shi, Zikai Wen, and Charalampos Papamanthou. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In *2016 IEEE symposium on security and privacy (SP)*, pages 839–858. IEEE, 2016.

[3] Iddo Bentov, Ranjit Kumaresan, and Andrew Miller. Instantaneous decentralized poker. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 410–440. Springer, 2017.

[4] S. Chong, J. Guttman, A. Datta, A. Myers, B. Pierce, P. Schaumont, T. Sherwood, and N. Zeldovich. Report on the NSF workshop on formal methods for security. *arXiv preprint arXiv:1608.00678*, 2016.

[5] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proceedings of the Symposium on Foundations of Computer Science (FOCS)*, 2001.

[6] Chris Hawblitzel, Jon Howell, Manos Kapritsos, Jacob R Lorch, Bryan Parno, Michael L Roberts, Srinath Setty, and Brian Zill. Ironfleet: proving practical distributed systems correct. In *Proceedings of the 25th Symposium on Operating Systems Principles*, pages 1–17. ACM, 2015.

[7] Karthikeyan Bhargavan, Barry Bond, Antoine Delignat-Lavaud, Cédric Fournet, Chris Hawblitzel, Catalin Hritcu, Samin Ishtiaq, Markulf Kohlweiss, Rustan Leino, Jay Lorch, et al. Everest: Towards a verified, drop-in replacement of https. In *2nd Summit on Advances in Programming Languages (SNAPL 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[8] Timo Hanke, Mahnush Movahedi, and Dominic Williams. Dfinity technology overview series, consensus system. *arXiv preprint arXiv:1805.04548*, 2018.

[9] Robin Milner. *Communicating and mobile systems: the pi calculus*. Cambridge university press, 1999.

[10] Ran Canetti, Asaf Cohen, and Yehuda Lindell. A simpler variant of universally composable security for standard multiparty computation. In *Annual Cryptology Conference*, pages 3–22. Springer, 2015.

[11] Kevin Liao, Matthew A. Hammer, and Andrew Miller. ILC: A calculus for composable, computational cryptography. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, 2019.

[12] Gilles Barthe, Juan Manuel Crespo, and César Kunz. Relational verification using product programs. In *International Symposium on Formal Methods*, pages 200–214. Springer, 2011.

[13] G. Barthe, B. Grégoire, S. Heraud, and S. Béguelin. Computer-aided security proofs for the working cryptographer. In *Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, 2011.

[14] Andrew Chi-Chih Yao. Protocols for secure computations. In *FOCS*, volume 82, pages 160–164, 1982.

[15] Bruno Barras, Samuel Boutin, Cristina Cornes, Judicael Courant, Jean-Christophe Filliatre, Eduardo Gimenez, Hugo Herbelin, Gerard Huet, Cesar Munoz, Chetan Murthy, et al. *The Coq proof assistant reference manual: Version 6.1*. PhD thesis, Inria, 1997.

[16] N. Swamy, C. Hriţcu, C. Keller, A. Rastogi, A. Delignat-Lavaud, S. Forest, K. Bhargavan, C. Fournet, et al. Dependent types and multi-monadic effects in F*. In *Proceedings of the Symposium on Principles of Programming Languages (POPL)*, 2016.

[17] Birgit Pfitzmann and Michael Waidner. A model for asynchronous reactive systems and its application to secure message transmission. In *Security and Privacy, 2001. S&P 2001. Proceedings. 2001 IEEE Symposium on*, pages 184–200. IEEE, 2001.

[18] Ran Canetti and Tal Rabin. Universal composition with joint state. In *Annual International Cryptology Conference*, pages 265–281. Springer, 2003.

[19] Michael Backes, Birgit Pfitzmann, and Michael Waidner. The reactive simulatability (rsim) framework for asynchronous systems. *Information and Computation*, 205(12):1685–1720, 2007.

[20] Ran Canetti, Ling Cheung, Dilsun Kaynar, Moses Liskov, Nancy Lynch, Olivier Pereira, and Roberto Segala. Analyzing security protocols using time-bounded task-pioas. *Discrete Event Dynamic Systems*, 18(1):111–159, 2008.

[21] Ueli Maurer and Renato Renner. Abstract cryptography. In *In Innovations in Computer Science*. Citeseer, 2011.

[22] Ueli Maurer. Constructive cryptography–a new paradigm for security definitions and proofs. In *Theory of Security and Applications*, pages 33–56. Springer, 2011.

[23] Florian Böhl and Dominique Unruh. Symbolic universal composability. *Journal of Computer Security*, 24(1):1–38, 2016.

[24] Dennis Hofheinz and Victor Shoup. Gnuc: A new universal composability framework. *Journal of Cryptology*, 28(3):423–508, 2015.

[25] Jan Camenisch, Robert R Enderlein, Stephan Krenn, Ralf Küsters, and Daniel Rausch. Universal composition with responsive environments. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 807–840. Springer, 2016.

[26] Jan Camenisch, Stephan Krenn, Ralf Küsters, and Daniel Rausch. iuc: Flexible universal composability made simple (full version).

[27] Jan Camenisch, Manu Drijvers, and Björn Tackmann. Multi-protocol uc and its use for building modular and efficient protocols.

[28] Martın Abadi and Phillip Rogaway. Reconciling two views of cryptography. In *Proceedings of the IFIP International Conference on Theoretical Computer Science*, pages 3–22. Springer, 2000.

[29] Melissa Chase and Anna Lysyanskaya. On signatures of knowledge. In *Annual International Cryptology Conference*, pages 78–96. Springer, 2006.

[30] Jonathan Protzenko, Jean-Karim Zinzindohoué, Aseem Rastogi, Tahina Ramananandro, Peng Wang, Santiago Zanella-Béguelin, Antoine Delignat-Lavaud, Cătălin Hriţcu, Karthikeyan Bhargavan, Cédric Fournet, et al. Verified low-level programming embedded in f. *Proceedings of the ACM on Programming Languages*, 1(ICFP):17, 2017.

[31] Xavier Leroy. Formal verification of a realistic compiler. *Communications of the ACM*, 52(7):107–115, 2009.

[32] Aymeric Fromherz, Nick Giannarakis, Chris Hawblitzel, Bryan Parno, Aseem Rastogi, and Nikhil Swamy. A verified, efficient embedding of a verifiable assembly language. *Proceedings of the ACM on Programming Languages*, 3(POPL):63, 2019.