Directed Study Report: NIZK Circuit Synthesis

Wyatt Howe
whowe@bu.edu

Professor Mayank Varia
varia@bu.edu

July 5, 2021

Abstract

Secure multi-party computation (MPC) allows mutually-distrusting parties to learn joint information about their secrets. MPC-in-the-head, however, allows for a single party to create a publicly-verifiable proof of a secret it knows. These two techniques can be composed to allow multiple distrusting parties to compute a publicly-verifiable proof about some property of their joint secrets. One application is setup as follows: Users contribute data to a primary server and secondary server in the form of XOR secret shares. An analyst may ask these servers to run a preapproved circuit on this private data. This is done within an ordinary two-party garbling except to prevent the servers from learning the output, the label mappings are decided at the beginning by the analyst. (So far this need be only an ordinary garbling.) But now, given our new means of evaluating *proof circuits*, we may additionally ensure the correctness of this execution (*e.g.* that the servers did not dishonestly modify the circuit) by having them run the garbling protocol on a version of the original circuit which has been transformed in order to also produce a (simulated MPC-in-the-head) proof.

1 Synthesis of Proof Circuits

Define the transformation $\mathcal{M}$ such that if C is a Boolean circuit computing the function $f : X \to Y$ where $C(x) = f(x)$, then $\mathcal{M}C$ is a Boolean circuit computing $F : X \to Y \times P$ where $(\mathcal{M}C)(x) = \langle f(x), p_{f(x)} \rangle$ and $p_{f(x)}$ is a publicly verifiable proof of that $f(x)$ was evaluated correctly.

As part of this project, we present a circuit synthesis tool capable of preforming this transformation. It is already possible to construct efficient NIZK proof $p_{f(x)}$ from an execution of f in non-interactive MPC-in-the-head. Semi-honest MPC on a function f with worst case executing cost n, will cost $O(n)$ to evaluate jointly. Suppose for our security parameter we will need K iterations of MPC in the head before computing the challenge, which is done once and takes time C. Our complexity is then raised to $K * O(n) + C$. Note that the oblivious if-else operations in the cut-and-choose step to create the proof incur an additional cost of $K * n$ of which we omit (because $K * n = o(C)$). The precise run-time is discussed at the end of this section.

1.1 Synthesis Technique

In order to write a tool for transforming a circuit, the tool must first be able to read the appropriate formats. For us, these are the 'Bristol Format' and 'Bristol Fashion[1]' circuit descriptions for Boolean

[1] https://homes.esat.kuleuven.be/~nsmart/MPC/index.html

circuits. Because the run time of the synthesis does not affect the resulting circuit, we chose to use Python which can load Bristol circuits into a `circuit` object, manipulate them gate-by-gate, and then emit as the same format. To synthesize existing functions as Boolean circuits, we use an embedded domain-specific language[13] as such: (1) mark all inputs as bits (2) keep track of which bits are operated on each other or constants (by overloading bit-wise operators in Python for the bits class) and (3) topologically sort the final ordering into a list of gates and emitted. With this technique, any Python function operating on bits can be automatically synthesized as a Boolean circuit.

1.2 MPC-in-the-head

Now, we take it a step further, and synthesize a function, E, specifically written to emulate an existing function, F. This existing function F can be read from a Bristol circuit $\langle F \rangle$, and thus, by synthesizing $E(F)$, we get a circuit identical to $\langle F \rangle$. What MPC-in-the-head does is effectively a gate-by-gate simulation, and so by modifying the way F emulates all AND, XOR, OR, NOT gates, we can build an $E'(F)$ circuit which runs MPC-in-the-head for F.

1.2.1 Beaver Triples

Because MPC-in-the-head is evaluated by only one party, we do not need its internal protocol to be maliciously secure, but instead at minimum only semi-honest[10]. This allows for us to reprocess all of the multiplications (same as AND-gates in $GF(2)$) done during the protocol by computing $(a, b, a * b)$ and secret sharing these three components to each simulated party.

Sharing and reconstructing secrets does not consume any triples or AND gates. Other nonlinear gates, such as OR gates, can be substituted by an emulated AND with two emulated XOR gates. Emulated XOR gates are simply pair-wise (local) XOR gates of each simulated party's share and have a negligible effect on the run-time, in the same way that real, garbled XOR gates in a garbled circuit would.

1.2.2 Cut-and-choose

To simulate a random oracle, we took the encryption function from the LowMC family of block ciphers, and adapted the reference code[2] to comply with our EDSL used in synthesis[3]. It is run in counter mode with a fixed key. Whenever the views accumulate more than the block size, 128 bits, the circuit computes `choices = choices XOR encrypt(views)` and the bits in the final `choices` block are used as choice bits for selecting which views get revealed as the proof.

1.2.3 Proof Size

In order for the circuit consume the right number of views while computing the challenge, it must be able to accurately estimate the total number of views generated. For N simulated parties and M AND-gates, we can expect a total of $M * (2N + 5) * (N - 1) * K$ views to get generated over all K repetitions. Naively optimizing for $N * M * K$ would dictate 5 or 6 parties as the optimal value of N, but here, we see that N should be minimized at the expense of K, and so $N = 3$ is in fact optimal. With the optimization of only showing one 'seed' view and its residual effects, the factor of $(N - 1)$ would be reduced to almost 1, thereby bringing the proof size down to almost

[2] `https://github.com/LowMC/lowmc/blob/master/LowMC.cpp`

[3] The authors could not find any other Boolean circuit description of LowMC, so ours may be the first.

$M * (2N + 5) * K$. The $(2N + 5)$ term is dependent on the triple multiplication procedure, but would likewise be optimizable with a similar strategy.

1.3 Cost of Evaluation

Computing a Beaver triple takes one multiplication. This cannot be improved or else the entire circuit would be completely evaluated during preprocessing, which is a contradiction. Computing one *emulated* AND gate consumes one Beaver triple, and requires $n + 2$ real/garbled AND gates.[4] At k repetitions of MPC-in-the-head totals to $k * (n + 3)$ AND gates in the proof circuit for every one AND gate in the original circuit. With $n = 5$ parties and $k = 40$ (which provides around 128-bits of security), this constant overhead factor would be 320x.

2 An Application to Publicly Verifiable MPC

One application for publicly verifiable proof circuits involves four types of parties: many data contributors, one or more data servers, two or more compute servers, and one or more analysts.

2.1 Server-Analyst Architecture

The server and analyst already know of a Boolean circuit C capable of computing the function $f : X \to Y$. They may define a multi-input function F as $F : X^n \to Y$ where $F(x_0, ..., x_n) = f(\bigoplus x_i)$. One of the x_i (the one known by the analyst) should be fixed among all $x \in X$, and the other(s) should be held by the data server or data servers. These can be XOR secret shares, and the privacy-preserving mechanism for submission, used by the data contributors, should take this structure into account.

The server will run two computations in a garbled circuit, one will use C to compute f, and one on $\mathcal{M}C$ to compute both F and p_F. If the lengths of the inputs are variable, then the circuit $\mathcal{M}C$ might need to be synthesized dynamically. Although the result of the first computation should be immediately available to the analyst, the publicly verifiable proof of F can potentially be published (or privately delivered to the analyst) at a later date (discussed more in section 2.3). If compute servers added to perform the garbling and evaluating on the behalf of the data server(s) and analyst(s), then the initial wire labels of each parties inputs should be chosen securely using oblivious transfer, such that the compute parties learn nothing but the final output—unless they collude. In the event that the compute parties would collude, all privacy is lost as it would be in any secret sharing based scheme. However, integrity of the circuit and proof is never compromised: the compute parties cannot forge an MPC-in-the-head proof using a different circuit, and therefore anything outputted from these servers is the result of the correct circuit on some input[5].

2.2 Composability

Because the 'proof' circuit synthesized by the tool is an ordinary Boolean circuit, it can be used by any existing garbling scheme. Authenticated garbling of these circuits is completely compatible, and would provide a additional security, for example, than an ordinary garbling.

[4] The cost is naively calculated as $2n+1$, but half of the $2n$ can be condensed into one, and so we get $n+1+1 = n+2$.

[5] If the circuit function isn't one-way, then the compute parties could still manipulate the output, just not the proof itself.

2.3 Deferring the Cost of Evaluating

The sample implementation linked below is set up to securely analyze the average (via a sum) and standard deviation (via a variance) of four 32-bit integers. The summation circuit, for example, costs 1 AND gate per adjacent bits (so $\ell - 1$) per sum for $m - 1$ sums. As referenced earlier, the cost per gate (in terms of AND gates) to emulate is $(n + 3) * k$. Thus, to generate a publicly verifiable proof within a garbled circuit, the garbler and evaluator would each have to process $(\ell - 1)(m - 1)(n + 3)k$ gates. For the parameters in the prototype, $k = 80$ rounds, $n = 5$ simulated MPC-in-the-head parties, $\ell = 4$ data points, $m = 32$, for bit arithmetic. This totals to 59,520 gates (excluding XOR/NOT/XNOR) which is very close to the cost of SHA-512. If we would lower the security parameter to 128-bits and let $k = 40$, then the cost would be 29,760 now which is a bit higher[6] than that of SHA-256.

3 Open Source Prototype

This project employs existing software libraries such as JIGG (for garbling) and `circuitry` (for synthesis) in order to implement the synthesis tool proposed usage in a two-party garbled evaluation setting. The open source repository containing the code written in this project is located at `https://github.com/wyatt-howe/nizk-proof-synthesis` and will likely continue to be updated in accordance with future work.

4 Future Work

4.1 Linking Security Models

If there were a way to link an execution of one protocol, semi-honest or otherwise, to an execution of a maliciously secure variant, even at a very high cost, then it would be possible to defer the execution of the proof circuit by the compute parties until they are either suspected of lying on the first execution, or a customer simply wants to order a proof. Would a solution in the server-analyst application generalize to a larger set of protocols? Note that this is only non-trivial when there are multiple parties involved.

4.2 Proofs of Arithmetic Circuits

Beaver triples, fixed-key block ciphers, garbled circuits, and MPC-in-the-head in general all support operating in $GF(n)$ rather than $GF(2)$. We would need to write a synthesis tool capable of reading arithmetic circuit descriptions, adapt JIGG for arithmetic circuits, and find an arithmetic-circuit-based block cipher. Thankfully, such block ciphers are already in development, such as MiMC[1] which takes advantage of $GF(n)$ and would be very suitable for an arithmetic variant of this paper.

4.3 Privacy Upon Collusion

It remains to be determined whether it is possible to preserve both integrity of the circuit *and* privacy of the inputs upon collusion. Circuit soldering is one avenue of investigation. Another idea is to try to reduce a privacy-safe scheme to homomorphic encryption in order to assert its likelihood.

[6] SHA-256 and 512 require 22,573 and 22,573 AND gates respectively and take around 2-10 minutes to garble and evaluate in a single-threaded web browser.

References

[1] M. Albrecht, L. Grassi, C. Rechberger, A. Roy, and T. Tiessen. Mimc: Efficient encryption and cryptographic hashing with minimal multiplicative complexity. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 191–219. Springer, 2016.

[2] C. Baum, I. Damgärd, and C. Orlandi. Publicly auditable secure multi-party computation. In *International Conference on Security and Cryptography for Networks*, pages 175–196. Springer, 2014.

[3] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*, pages 351–371. 2019.

[4] M. Chase, D. Derler, S. Goldfeder, C. Orlandi, S. Ramacher, C. Rechberger, D. Slamanig, and G. Zaverucha. Post-quantum zero-knowledge and signatures from symmetric-key primitives. In *Proceedings of the 2017 acm sigsac conference on computer and communications security*, pages 1825–1842, 2017.

[5] R. Cunningham, B. Fuller, and S. Yakoubov. Catching mpc cheaters: Identification and openability. In *International Conference on Information Theoretic Security*, pages 110–134. Springer, 2017.

[6] I. Damgärd, K. G. Larsen, and J. B. Nielsen. Communication lower bounds for statistically secure mpc, with or without preprocessing. In *Annual International Cryptology Conference*, pages 61–84. Springer, 2019.

[7] I. Giacomelli, J. Madsen, and C. Orlandi. Zkboo: Faster zero-knowledge for boolean circuits. In *25th {usenix} security symposium ({usenix} security 16)*, pages 1069–1083, 2016.

[8] Y. Gvili, J. Ha, S. Scheffler, M. Varia, Z. Yang, and X. Zhang. Turboikos: Improved non-interactive zero knowledge and post-quantum signatures.

[9] C. Hong, J. Katz, V. Kolesnikov, W.-j. Lu, and X. Wang. Covert security with public verifiability: faster, leaner, and simpler. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 97–121. Springer, 2019.

[10] Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai. Zero-knowledge from secure multiparty computation. In *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, pages 21–30, 2007.

[11] M. Jawurek, F. Kerschbaum, and C. Orlandi. Zero-knowledge using garbled circuits.

[12] J. Katz, V. Kolesnikov, and X. Wang. Improved non-interactive zero knowledge with applications to post-quantum signatures. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 525–537, 2018.

[13] A. Lapets, W. Howe, B. Getchell, and F. Jansen. An embedded domain-specific language for logical circuit descriptions with applications to garbled circuits. *IACR Cryptol. ePrint Arch.*, 2020:1604, 2020.

[14] A. Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[15] A. C. Yao. Protocols for Secure Computations. In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*, SFCS '82, pages 160–164, Washington, DC, USA, 1982. IEEE Computer Society.