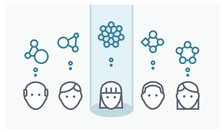



PACCMulE: Private Aggregate Circuit Commitments for Multiple Entities

Dan Boneh Uma Dayal Bharath Namboothiry Wilson Nguyen Alex Ozdemir

A Friendly Competition

Imagine a Kaggle competition with n competitors and a public scoring algorithm. The competitors would like to participate and have a winner declared, but would like to keep their proprietary algorithms and output secret. How can we provide this cryptographic functionality?



Formalization: N-Party Private Circuits

To formalize the problem, we introduce a class of arithmetic circuits known as n-party private, or $\mathcal{NPP}$. A circuit A is in $\mathcal{NPP}$ if it contains at most n disjoint subcircuits $A_1 \dots A_n$ such that each A_i is known only to party i, all of which feed to a public connecting subcircuit A_p.

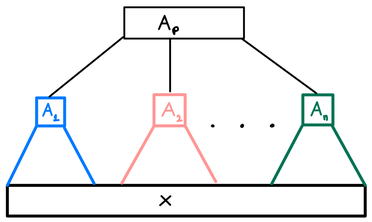


Figure 1. Example N-Party Private Circuit Representation

Here, we can think of each private sub-circuit as a competing algorithm and the public connector as a common scoring algorithm.

Circuit Commitment Schemes

Traditionally, a circuit commitment between a prover $\mathcal{P}$ with a circuit A and a verifier $\mathcal{V}$ works as follows

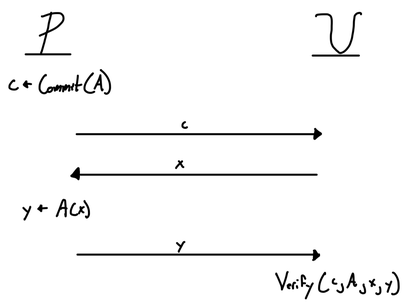


Figure 2. Circuit Commitment Scheme Depiction

There are a number of working commitment schemes, each with unique pros/cons. In our work, we center ourselves around the methods used in the Plonk Commitment scheme [3].

What is PACCMulE?

We are trying to create an analogous commitment scheme in this multi-entity setting. Formally, we define a **PACCMulE** for a n-party private circuit $A = \bigcup_{i \in [n] \cup \{p\}} A_i$ is a tuple (Setup, Commit, Eval) where:

- **Setup**$(1^\lambda, \{N_1 \dots, N_n, N_p\}) \to$ pp: Sample public parameters.
- **Commit**$(pp, A \in AC, r \in R) \to c \in \mathcal{C}$: Produce a commitment of A.
- **Eval**$(\mathcal{P}_1(\text{pp}, A_1, A_p, r \in \text{R}, x \in \mathcal{X}, y \in \mathcal{Y}), \dots \mathcal{P}_n(\text{pp}, A_n, A_p, r \in \text{R}, x \in \mathcal{X}, y \in \mathcal{Y}), \mathcal{V}(\text{pp}, c, x, y)) \to \{0, 1\}$: Convince all parties that $A(x) = y$

Plonk Encoding

The Plonk scheme maps the pins of a circuit to a multiplicative groups $\mathbb{K} = \langle \gamma \rangle$ with order divisible by 3, and maps the gates to and $\mathbb{K}_g = \langle \gamma^3 \rangle$.

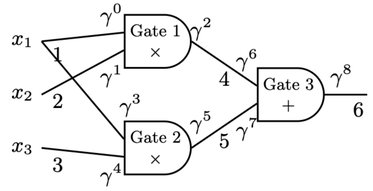


Figure 3. Plonk Encoded Circuit

A circuit A is then defined with the following polynomials, which are interpolated using Fast Fourier Transform (FFT):

- *Selector Polynomial*: $s(g \in \mathbb{K}_g)$, returns 1 if gate $g = +$ and 0 otherwise
- *Wiring Polynomial*: $w(k \in K)$ returns the pin that i is wired to

When A is run on an input x, a computation trace can be performed to determine the values at each pin in $\mathbb{K}$. The prover will interpolate these values into a *Value Polynomial* p.

Proof of Functional Relation

In settings where a committed circuit is not fully known to the verifier, such as [1], the verifier would like to be assured that what is being committed to is a function. In Plonk, this means convincing the verifier the following hold for a committed circuit $A = (w, s)$:

1. *Permutation Check:* w is a permutation
2. *Selector Check:* s is binary over $\mathbb{K}_g$
3. *Representative Check:* The Union of the initial input pins and all gate output pins is a representative set of $\overline{W}$, the partition induced by w
4. *Topological Sort:* $\overline{W}$ can be topologically sorted

PACCMulE Objectives

Stated plainly, a PACCMulE is circuit commitment scheme with PFR for any $A \in \mathcal{NPP}$. Concretely this involves constructing a convincing proof of the following claims for each $j \in [n] \cap \{p\}$:

- Ensure that $A \in \mathcal{NPP}$
- Ensure that A_j is committed correctly
- Make additional checks for the PFR on A
- Confirm that $A(x) = y$

Desired Security Properties

A secure PACCMulE has the following properties:

- **Committing**: The tuple (Setup, Commit) is a hiding and binding commitment scheme for the function space $\mathcal{NPP}$.
- **Complete Eval**: Eval will accept any (A, x, y, c, r), that satisfy the relation $\mathcal{R}_{\text{eval}}$, which demands that:

$$A \in \mathcal{NPP} \quad \wedge \quad A(x) = y \quad \wedge \quad c = \text{Commit}(pp, A, r)$$

- **Extractable** Eval is an argument of knowledge for the relation $R_{\text{eval}}(pp)$.
- **Honest Verifier** n**-Zero-Knowledge**: Up to n-colluding dishonest provers learn nothing about the components of the other provers, other than the validity of A.

Polynomial Interactive Oracle Proofs

A *Polynomial Interactive Oracle Proof*, or PolyIOP, is an interactive protocol where a prover has the power to grant polynomial oracle access to a verifier. A polyIOP for a relation is secure if it has the following properties:

1. **Completeness**: If an input satisfies the , then the protocol accepts
2. **Soundness**: If an input does not satisfy the , then the protocol rejects with negligible error
3. **Honest Verifier Zero Knowledge**: A verifier learns nothing more of prover secrets than the output of the protocol

It will suffice to construct a secure polyIOP that captures the PACCMulE objectives. Then, we can make use of some handy derandomization techniques [2] to compile the protocol into a single, non-interactive proof.

Constructed Protocols on Arbitrary Sets

To create a secure polyIOP for PACCMulE, we first construct the following secure protocols. A large part of our contribution was getting these to work over any subset $I \subseteq \mathbb{K}$, where I may not have any clean structure to take advantage of.

- **Zero:** Check if a polynomial $f(i) = 0$ for all $i \in I$
- **Non-Zero:** Check if a polynomial $f(i) \neq 0$ for all $i \in I$
- **Set Product:** Check if the product of a polynomial over I is equal to
- **Generalized Multiset Equality:** Images of two polynomials are the same as multisets over an arbitrary subset I of subgroup $\mathbb{K}$
- **Permutation:** Check that a polynomial is a permutation over I
- **Permutation Composition:** Check that two polynomials are equal on cycles of a permutation
- **Geometric Sequence Test:** Check that a polynomial has a geometric progression
- **Representative Check:** Perform the representative check of PFR on the subcircuit defined over I
- **Expanded Discrete Log Comparison:** Check that the discrete log of a polynomial is greater than another across all of I
- **Topological Sort:** Verify that the circuit can be topologically sorted

After constructing the above secure protocols, we are able to deploy them and create a polyIOP for PACCMulE!

Implications on State of the Art

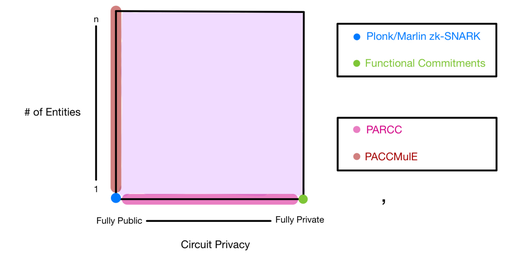
Reduction to $\mathcal{NPP}$

In the general case, we may want to relax our multi-entity setting to any arithmetic circuit comprised of at most n disjoint privately held sub-circuits. As it turns out, circuits like these can be perfectly partitioned into subcircuits $\in \mathcal{NPP}$. So, PACCMuLE is able to provide functionality in the general setting!

Broadening the State of the Art

The current state of the art circuit commitment schemes achieve functionality for public circuits [3] and fully private circuits via PFR [1].

In our preceding project, **PARCC: Partially Revealable Circuit Commitments**, we broaden the scope of this work to circuits that contain any combination of public and private sub-circuits, such that private components remain hidden while public components are revealed. In this project we work along the other axis, creating commitment schemes for any number of private proving entities.



Combining both of our summer research projects, we have exponentially grown the class of circuits that can be committed to!

Future Research Directions

Shared Library Support

Arithmetic circuits have no way to compress repeating or shared architecture. One goal is to a commitment scheme that supports shared use, which would have many real-world use cases.

Supporting Marlin Encoding

Marlin is another circuit commitment scheme with a much more elegant PFR. Another goal of ours is to develop analogous PARCC and PACCMulE schemes from Marlin, which may lead to runtime and/or proof size savings.

References

[1] Dan Boneh, Wilson Nguyen, and Alex Ozdemir.
Efficient functional commitments: How to commit to a private function.
Cryptology ePrint Archive, Paper 2021/1342, 2021.
`https://eprint.iacr.org/2021/1342`.

[2] Amos Fiat and Adi Shamir.
How to prove yourself: Practical solutions to identification and signature problems.
In Andrew M. Odlyzko, editor, *Advances in Cryptology — CRYPTO' 86*, pages 186–194, Berlin, Heidelberg, 1987. Springer Berlin Heidelberg.

[3] Ariel Gabizon, Zachary J. Williamson, and Oana-Madalina Ciobotaru.
Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge.
IACR Cryptol. ePrint Arch., 2019:953, 2019.