

Investigating IVC with Halo2

Rasmus Kirk Jakobsen - 201907084

2024-12-16 - 16:36:45 UTC

Contents

Introduction	1
Prerequisites	2
PC_{DL}: The Polynomial Commitment Scheme	2
Outline	3
PC _{DL} .COMMIT	3
PC _{DL} .OPEN	4
PC _{DL} .SUCCINCTCHECK	5
PC _{DL} .CHECK	5
Completeness	5
Soundness	6
Zero-knowledge	6
Benchmarks	6
AS_{DL}: The Accumulation Scheme	6
Common subroutine	6
Prover	6
Verifier	7
Decider	7
Completeness	7
Soundness	7
Zero-knowledge	7
Benchmarks	7
Appendix	8
Notation	8
CM: Pedersen Commitment	8

Introduction

Halo2, can be broken down into the following components:

- **Plonk**: A general-purpose zero-knowledge proof scheme.
- **PC_{DL}**: A Polynomial Commitment Scheme in the Discrete Log setting.
- **AS_{DL}**: An Accumulation Scheme in the Discrete Log setting.
- **Pasta**: A Cycle of Elliptic Curves, namely **Pallas** and **Vesta**.

This project is focused on the components of PC_{DL} and AS_{DL}. I used the [2020 paper](#) “*Proof-Carrying Data from Accumulation Schemes*” as a reference. The project covers both the theoretical aspects of the scheme described in this document along with a rust implementation, both of which can be found in the project’s [repository](#).

The original paper had additional functions for generating prover and verifier keys, getting the public parameters and trimming input to fit the public parameters. I have chosen to omit these from the discussion below as these are fixed per-implementation.

Prerequisites

Basic knowledge on elliptic curves, groups, interactive arguments are assumed in the following text. There is also a heavy reliance on the Inner Product Proof from the Bulletproofs protocol, see the following resources on bulletproofs if need be:

- [Section 3 in the original Bulletproofs paper](#)
- [From Zero \(Knowledge\) to Bulletproofs writeup](#)
- [Rust Dalek Bulletproofs implementation notes](#)
- [Section 4.1 of my bachelors thesis](#)

PC_{DL}: The Polynomial Commitment Scheme

$$\begin{aligned}
C_{i+1} &= \langle \mathbf{c}_{i+1}, \mathbf{G}_{i+1} \rangle + \langle \mathbf{c}_{i+1}, \mathbf{z}_{i+1} \rangle H' \\
&= \langle l(\mathbf{c}_i) + \xi_{i+1}^{-1} r(\mathbf{c}_i), l(\mathbf{G}_i) + \xi_{i+1} r(\mathbf{G}_i) \rangle + \langle l(\mathbf{c}_i) + \xi_{i+1}^{-1} r(\mathbf{c}_i), l(\mathbf{z}_i) + \xi_{i+1} r(\mathbf{z}_i) \rangle H' \\
&= \langle l(\mathbf{c}_i), l(\mathbf{G}_i) \rangle + \xi_{i+1} \langle l(\mathbf{c}_i), r(\mathbf{G}_i) \rangle + \xi_{i+1}^{-1} \langle r(\mathbf{c}_i), l(\mathbf{G}_i) \rangle + \langle r(\mathbf{c}_i), r(\mathbf{G}_i) \rangle \\
&\quad + (\langle l(\mathbf{c}_i), l(\mathbf{z}_i) \rangle + \xi_{i+1} \langle l(\mathbf{c}_i), r(\mathbf{z}_i) \rangle + \xi_{i+1}^{-1} \langle r(\mathbf{c}_i), l(\mathbf{z}_i) \rangle + \langle r(\mathbf{c}_i), l(\mathbf{z}_i) \rangle) H'
\end{aligned}$$

We can further group these terms:

$$\begin{aligned}
C_{i+1} &= \langle l(\mathbf{c}_i), l(\mathbf{z}_i) \rangle + \langle r(\mathbf{c}_i), r(\mathbf{G}_i) \rangle + \xi_{i+1} \langle l(\mathbf{c}_i), r(\mathbf{G}_i) \rangle + \xi_{i+1}^{-1} \langle r(\mathbf{c}_i), l(\mathbf{G}_i) \rangle \\
&\quad + (\langle l(\mathbf{c}_i), l(\mathbf{z}_i) \rangle + \langle r(\mathbf{c}_i), r(\mathbf{z}_i) \rangle) H' + \xi_{i+1} \langle l(\mathbf{c}_i), r(\mathbf{z}_i) \rangle H' + \xi_{i+1}^{-1} \langle r(\mathbf{c}_i), l(\mathbf{z}_i) \rangle H' \\
&= C_i + \xi_{i+1} R_i + \xi_{i+1}^{-1} L_i
\end{aligned}$$

Where:

$$\begin{aligned}
L_i &= \langle r(\mathbf{c}_i), l(\mathbf{G}_i) \rangle + \langle r(\mathbf{c}_i), l(\mathbf{z}_i) \rangle H' \\
R_i &= \langle l(\mathbf{c}_i), r(\mathbf{G}_i) \rangle + \langle l(\mathbf{c}_i), r(\mathbf{z}_i) \rangle H'
\end{aligned}$$

And then simplify further:

$$\begin{aligned}
\mathbf{L} &= (L_0, \dots, L_{\lg(n)-1}) \\
\mathbf{R} &= (R_0, \dots, R_{\lg(n)-1}) \\
\mathbf{C} &= (C_0, \dots, C_{\lg(n)}) \\
\boldsymbol{\xi} &= (\xi_0, \dots, \xi_{\lg(n)})
\end{aligned}$$

Now we are ready to look at the check that the verifier makes:

$$\begin{aligned}
C_0 &= \bar{C} + vH' = C + vH' \\
C_{\lg(n)} &= C_0 + \sum_{i=0}^{\lg(n)-1} \xi_{i+1}^{-1} L_i + \xi_{i+1} R_i
\end{aligned}$$

The original definition of C_i :

$$= \langle \mathbf{c}_{\lg(n)}, \mathbf{G}_{\lg(n)} \rangle + \langle \mathbf{c}_{\lg(n)}, \mathbf{z}_{\lg(n)} \rangle H'$$

Vectors have length one, use the single elements $c^{(0)}, G^{(0)}, c^{(0)}, z^{(0)}$:

$$= c^{(0)} G^{(0)} + c^{(0)} z^{(0)} H'$$

The verifier has $c^{(0)} = c, G^{(0)} = U$ from $\pi \in \mathbf{EvalProof}$:

$$= cU + cz^{(0)} H'$$

And finally, by construction of $h(X) \in \mathbb{F}_q^d[X]$

$$= cU + ch(z)H'$$

Which corresponds exactly to the check that the verifier makes.

Outline

We have four main functions:

- $\text{PC}_{\text{DL}}.\text{COMMIT}(p : \mathbb{F}_q^d[X], \omega : \mathbf{Option}(\mathbb{F}_q)) \rightarrow \mathbb{E}(\mathbb{F}_q)$:
Creates a commitment to the coefficients of the polynomial q of degree d with optional hiding ω , using pedersen commitments.
- $\text{PC}_{\text{DL}}.\text{OPEN}(p : \mathbb{F}_q^d[X], C : \mathbb{E}(\mathbb{F}_q), z : \mathbb{F}_q, \omega : \mathbf{Option}(\mathbb{F}_q)) \rightarrow \pi_{\text{EVAL}}$:
Creates a proof π that states: “I know $p \in \mathbb{F}_q^d[X]$ with commitment $C \in \mathbb{E}(\mathbb{F}_q)$ s.t. $p(z) = v$ ” where p is private and d, z, v are public.
- $\text{PC}_{\text{DL}}.\text{SUCCINCTCHECK}(C : \mathbb{E}(\mathbb{F}_q), d : \mathbb{N}, z : \mathbb{F}_q, v : \mathbb{F}_q, \pi : \pi_{\text{EVAL}}) \rightarrow \mathbf{Result}(\mathbb{F}_q^d[X], \mathbb{G})$:
Cheaply checks that a proof π is correct. It is not a full check however, since an expensive part of the check is deferred until a later point.
- $\text{PC}_{\text{DL}}.\text{CHECK}(C : \mathbb{E}(\mathbb{F}_q), d : \mathbb{N}, z : \mathbb{F}_q, v : \mathbb{F}_q, \pi : \pi_{\text{EVAL}}) \rightarrow \mathbf{Result}(\top, \perp)$:
The full check on π .

$\text{PC}_{\text{DL}}.\text{Commit}$

$\text{PC}_{\text{DL}}.\text{COMMIT}$ is rather simple, we just take the coefficients of the polynomial and commit to them using a pedersen commitment:

Algorithm 1 $\text{PC}_{\text{DL}}.\text{COMMIT}$

Inputs

- | | |
|--|--|
| $p : \mathbb{F}_q^d[X]$ | The univariate polynomial that we wish to commit to. |
| $\omega : \mathbf{Option}(\mathbb{F}_q)$ | Optional hiding factor for the commitment. |

Output

- | | |
|--------------------------------|---|
| $C : \mathbb{E}(\mathbb{F}_q)$ | The pedersen commitment to the coefficients of polynomial p . |
|--------------------------------|---|

Require: $d \leq D$

Require: $(d + 1) = 2^k$, where $k \in \mathbb{N}$

- 1: Let \mathbf{p} be the coefficient vector for p
 - 2: Output $C := \text{CM}.\text{COMMIT}(\mathbf{G}, \mathbf{p}, \omega)$.
-

PC_{DL}.Open

Algorithm 2 PC_{DL}.OPEN

Inputs

$p : \mathbb{F}_q^d[X]$	The univariate polynomial that we wish to open for.
$C : \mathbb{E}(\mathbb{F}_q)$	A commitment to the coefficients of p .
$z : \mathbb{F}_q$	The element that z will be evaluated on $v = p(z)$.
$\omega : \text{Option}(\mathbb{F}_q)$	Optional hiding factor for C . <i>Must</i> be included if C was created with hiding!

Output

EvalProof	Proof that states: "I know $p \in \mathbb{F}_q^d[X]$ with commitment $C \in \mathbb{E}(\mathbb{F}_q)$ s.t. $p(z) = v$ "
------------------	---

Require: $d \leq D$

Require: $(d+1) = 2^k$, where $k \in \mathbb{N}$

- 1: Compute $v = p(z)$ and let $n = d+1$.
 - 2: Sample a random polynomial $\bar{p} \in \mathbb{F}_q^{\leq d}[X]$ such that $\bar{p}(z) = 0$.
 - 3: Sample corresponding commitment randomness $\bar{\omega} \in \mathbb{F}_q$.
 - 4: Compute a hiding commitment to \bar{p} : $\bar{C} \leftarrow \text{CM.COMMIT}(\mathbf{G}, \bar{p}, \bar{\omega}) \in \mathbb{G}$.
 - 5: Compute the challenge $\alpha := \rho_0(C, z, v, \bar{C}) \in \mathbb{F}_q^*$.
 - 6: Compute the polynomial $p' := p + \alpha \bar{p} = \sum_{i=0}^d c_i X_i \in \mathbb{F}_q[X]$.
 - 7: Compute commitment randomness $\omega' := \omega + \alpha \bar{\omega} \in \mathbb{F}_q$.
 - 8: Compute a non-hiding commitment to p' : $C' := C + \alpha \bar{C} - \omega' S \in \mathbb{G}$.
 - 9: Compute the 0-th challenge field element $\xi_0 := \rho_0(C', z, v) \in \mathbb{F}_q$, then $H' := \xi_0 H \in \mathbb{G}$.
 - 10: Initialize the vectors (\mathbf{c}_0 is defined to be coefficient vector of p'):

$$\mathbf{c}_0 := (c_0, c_1, \dots, c_d) \in F_q^n$$

$$\mathbf{z}_0 := (1, z^1, \dots, z^d) \in F_q^n$$

$$\mathbf{G}_0 := (G_0, G_1, \dots, G_d) \in \mathbb{G}_n$$
 - 11: **for** $i \in [\lg(n)]$ **do**
 - 12: Compute $L_i := \text{CM.COMMIT}(l(\mathbf{G}_{i-1}) \oplus H', r(\mathbf{c}_{i-1}) + \langle r(\mathbf{c}_{i-1}), l(\mathbf{z}_{i-1}) \rangle, \perp)$
 - 13: Compute $R_i := \text{CM.COMMIT}(r(\mathbf{G}_{i-1}) \oplus H', l(\mathbf{c}_{i-1}) + \langle l(\mathbf{c}_{i-1}), r(\mathbf{z}_{i-1}) \rangle, \perp)$
 - 14: Generate the i -th challenge $\xi_i := \rho_0(\xi_{i-1}, L_i, R_i) \in \mathbb{F}_q$.
 - 15: Construct commitment inputs for the next round:

$$\mathbf{G}_i := l(\mathbf{G}_{i-1}) + \xi_i \cdot r(\mathbf{G}_{i-1})$$

$$\mathbf{c}_i := l(\mathbf{c}_{i-1}) + \xi_i^{-1} \cdot r(\mathbf{c}_{i-1})$$

$$\mathbf{z}_i := l(\mathbf{z}_{i-1}) + \xi_i \cdot r(\mathbf{z}_{i-1})$$
 - 16: **end for**
 - 17: Finally output the evaluation proof $\pi := (\mathbf{L}, \mathbf{R}, U := \mathbf{G}_{\lg(n)}, c := \mathbf{c}_{\lg(n)}, \bar{C}, \omega')$
-

The PC_{DL}.OPEN algorithm simply follows the proving algorithm from bulletproofs. Except, in this case we are trying to prove we know polynomial p s.t. $v = \langle \mathbf{c}_0, \mathbf{z}_0 \rangle$. So because z is public, we can get away with omitting the generators for \mathbf{b} in the original protocol (\mathbf{H}).

PC_{DL}.SuccinctCheck

Algorithm 3 PC_{DL}.SUCCINCTCHECK

Inputs

$C : \mathbb{E}(\mathbb{F}_q)$	A commitment to the coefficients of p .
$d : \mathbb{N}$	The degree of p
$z : \mathbb{F}_q$	The element that p is evaluated on.
$v : \mathbb{F}_q$	The claimed element $v = p(z)$.
$\pi : \mathbf{EvalProof}$	The evaluation proof produced by PC _{DL} .OPEN

Output

Result $((\mathbb{F}_q^d[X], \mathbb{G}), \perp)$	The algorithm will either succeed and output $(h : \mathbb{F}_q^d[X], U : \mathbb{G})$ if π is a valid proof and otherwise fail (\perp) .
--	---

Require: $d \leq D$

Require: $(d + 1) = 2^k$, where $k \in \mathbb{N}$

- 1: Parse π as $(\mathbf{L}, \mathbf{R}, U := \mathbf{G}_{lg(n)}, c := c_{lg(n)}, \bar{C}, \omega')$ and let $n = d + 1$.
 - 2: **Compute the challenge** $\alpha := \rho_0(C, z, v, \bar{C}) \in F_q^*$.
 - 3: Compute the non-hiding commitment $C' := C + \alpha \bar{C} - \omega' S \in \mathbb{G}$.
 - 4: Compute the 0-th challenge: $\xi_0 := \rho_0(C', z, v)$, and set $H' := \xi_0 H \in \mathbb{G}$.
 - 5: Compute the group element $C_0 := C' + v H' \in \mathbb{G}$.
 - 6: **for** $i \in [lg(n)]$ **do**
 - 7: Generate the i-th challenge: $\xi_i := \rho_0(\xi_{i-1}, L_i, R_i) \in \mathbb{F}_q$.
 - 8: Compute the i-th commitment: $C_i := \xi_i^{-1} L_i + C_{i-1} + \xi_i R_i \in \mathbb{G}$.
 - 9: **end for**
 - 10: Define the univariate polynomial $h(X) := \prod_{i=0}^{lg(n)-1} (1 + \xi_{lg(n)-i} X^{2^i}) \in \mathbb{F}_q[X]$.
 - 11: Compute the evaluation $v' := c \cdot h(z) \in \mathbb{F}_q$.
 - 12: Check that $C_{lg(n)} \stackrel{?}{=} \text{CM.COMMIT}(U \# H', c \# v', \perp)$
 - 13: Output $(h(X), U)$.
-

PC_{DL}.Check

Algorithm 4 PC_{DL}.CHECK

Inputs

$C : \mathbb{E}(\mathbb{F}_q)$	A commitment to the coefficients of p .
$d : \mathbb{N}$	The degree of p
$z : \mathbb{F}_q$	The element that p is evaluated on.
$v : \mathbb{F}_q$	The claimed element $v = p(z)$.
$\pi : \mathbf{EvalProof}$	The evaluation proof produced by PC _{DL} .OPEN

Output

Result (\top, \perp)	The algorithm will either succeed (\top) if π is a valid proof and otherwise fail (\perp) .
-------------------------------	---

Require: $d \leq D$

Require: $(d + 1) = 2^k$, where $k \in \mathbb{N}$

- 1: Check that PC_{DL}.SUCCINCTCHECK(C, d, z, v, π) accepts and outputs (h, U) .
 - 2: Check that $U \stackrel{?}{=} \text{CM.COMMIT}(\mathbf{G}, \mathbf{h}, \perp)$, where \mathbf{h} is the coefficient vector of the polynomial h .
-

Completeness

This section will both function as an explainer of what is going on in this algorithm, along with a more formal proof of completeness.

Soundness

Zero-knowledge

Benchmarks

AS_{DL}: The Accumulation Scheme

Common subroutine

Algorithm 5 AS_{DL}.COMMONSUBROUTINE

Inputs

$d : \mathbb{N}$ The degree of p .
 $\mathbf{q} : \mathbb{F}_q^m$ New instances *and accumulators* to be accumulated.
 $\pi_V : \text{Option}(\text{AccHiding})$ Necessary parameters if hiding is desired.

Output

Result $((\mathbb{E}(\mathbb{F}_q), \mathbb{N}, \mathbb{F}_q, \mathbb{F}_q^d[X]), \perp)$ The algorithm will either succeed $(\mathbb{E}(\mathbb{F}_q), \mathbb{N}, \mathbb{F}_q, \mathbb{F}_q^d[X])$ if the instances has consistent degree and hiding parameters and otherwise fail (\perp) .

Require: $d \leq D$

Require: $(d+1) = 2^k$, where $k \in \mathbb{N}$

- 1: **for** $i \in [m]$ **do**
 - 2: Parse q_i as a tuple $((C_i, d_i, z_i, v_i), \pi_i)$.
 - 3: Compute $(h_i(X), U_i) := \text{PC}_{\text{DL}}.(C_i, z_i, v_i, \pi_i)$.
 - 4: Check that $d_i = D$ (We accumulate only the degree bound D .)
 - 5: **end for**
 - 6: Parse π_V as (h_0, U_0, ω) , where $h_0(X) = aX + b \in \mathbb{F}_q[X]$, $U_0 \in \mathbb{G}$ and $\omega \in \mathbb{F}_q$
 - 7: Check that U_0 is a deterministic commitment to h_0 : $U_0 = \text{PC}_{\text{DL}}.\text{COMMIT}(h, \perp)$.
 - 8: Compute the challenge $\alpha := \rho_1(\mathbf{h}, \mathbf{U}) \in \mathbb{F}_q$
 - 9: Let the polynomial $h(X) := \sum_{i=0}^m \alpha^i h_i \in \mathbb{F}_q[X]$
 - 10: Compute the accumulated commitment $C := \sum_{i=0}^m \alpha^i U_i$
 - 11: Compute the challenge $z := \rho_1(C, h) \in \mathbb{F}_q$.
 - 12: Randomize C : $\bar{C} := C + \omega S \in \mathbb{G}$.
 - 13: Output $(\bar{C}, d, z, h(X))$.
-

Prover

Algorithm 6 AS_{DL}.PROVER

Inputs

$d : \mathbb{N}$ The degree of p .
 $\mathbf{q} : \mathbb{F}_q^m$ New instances *and accumulators* to be accumulated.

Output

Result (Acc, \perp) The algorithm will either succeed $((\bar{C}, d, z, v, \pi), \pi_V) \in \text{Acc}$ if the instances has consistent degree and hiding parameters and otherwise fail (\perp) .

Require: $d \leq D$

Require: $(d+1) = 2^k$, where $k \in \mathbb{N}$

- 1: Let $n = d + 1$
 - 2: Sample a random linear polynomial $h_0 \in F_q[X]$
 - 3: Then compute a deterministic commitment to h_0 : $U_0 := \text{PC}_{\text{DL}}.\text{COMMIT}(h_0, \perp)$
 - 4: Sample commitment randomness $\omega \in F_q$, and set $\pi_V := (h_0, U_0, \omega)$.
 - 5: Then, compute the tuple $(\bar{C}, d, z, h(X)) := \text{AS}_{\text{DL}}.\text{COMMONSUBROUTINE}(d, \mathbf{q}, \pi_V)$.
 - 6: Compute the evaluation $v := h(z)$
 - 7: Generate the hiding evaluation proof $\pi := \text{PC}_{\text{DL}}.\text{OPEN}(h(X), \bar{C}, d, z, \omega)$.
 - 8: Finally, output the accumulator $\text{acc} = ((\bar{C}, d, z, v, \pi), \pi_V)$.
-

Verifier

Algorithm 7 $\text{AS}_{\text{DL}}.\text{VERIFIER}$

Inputs

$\mathbf{q} : \mathbb{F}_q^m$ New instances *and accumulators* to be accumulated.
 $\mathbf{acc} : \mathbf{Acc}$ The accumulator.

Output

Result(\top, \perp) The algorithm will either succeed (\top) if TODO and otherwise fail (\perp).

Require: $\mathbf{acc}.d \leq D$

Require: $(\mathbf{acc}.d + 1) = 2^k$, where $k \in \mathbb{N}$

- 1: Parse \mathbf{acc} as $((\bar{C}, d, z, v, _), \pi_V)$
 - 2: The accumulation verifier computes $(\bar{C}', d', z', h(X)) := \text{AS}_{\text{DL}}.\text{COMMONSUBROUTINE}(d, \mathbf{q}, \pi_V)$
 - 3: Then checks that $\bar{C}' \stackrel{?}{=} \bar{C}, d' \stackrel{?}{=} d, z' \stackrel{?}{=} z$, and $h(z) \stackrel{?}{=} v$.
-

Decider

Algorithm 8 $\text{AS}_{\text{DL}}.\text{DECIDER}$

Inputs

$\mathbf{acc} : \mathbf{Acc}$ The accumulator.

Output

Result(\top, \perp) The algorithm will either succeed (\top) if TODO and otherwise fail (\perp).

Require: $\mathbf{acc}.d \leq D$

Require: $(\mathbf{acc}.d + 1) = 2^k$, where $k \in \mathbb{N}$

- 1: Parse \mathbf{acc} as $((\bar{C}, d, z, v, \pi), _)$
 - 2: Check $\top \stackrel{?}{=} \text{PC}_{\text{DL}}.\text{CHECK}(\bar{C}, d, z, v, \pi)$
-

Completeness

Soundness

Zero-knowledge

Benchmarks

Appendix

Notation

$(\mathbb{F}_q, \dots, \mathbb{E}(\mathbb{F}_q))$: Same as $\mathbb{F}_q \times \dots \times \mathbb{E}(\mathbb{F}_q)$

$\langle \mathbf{a}, \mathbf{G} \rangle$ where $\mathbf{a} \in \mathbb{F}_q^n, \mathbf{G} \in \mathbb{E}^n(\mathbb{F}_q)$: The dot product of field elements \mathbf{a} and curve points \mathbf{G} ($\sum_{i=0}^n a_i G_i$).

$\langle \mathbf{a}, \mathbf{b} \rangle$ where $\mathbf{a} \in \mathbb{F}_q^n, \mathbf{b} \in \mathbb{F}_q^n$: The dot product of vectors \mathbf{a} and \mathbf{b} .

$l(\mathbf{a})$: Gets the left half of \mathbf{a} .

$r(\mathbf{a})$: Gets the right half of \mathbf{a} .

$\mathbf{a} \# \mathbf{b}$ where $\mathbf{a} \in \mathbb{F}_q^n, \mathbf{b} \in \mathbb{F}_q^m$: Concatenate vectors to create $\mathbf{c} \in \mathbb{F}_q^{n+m}$.

$a \# b$ where $a \in \mathbb{F}_q$: Create vector $\mathbf{c} = (a, b)$.

“Type aliases”

EvalProof = $(\mathbb{E}^{lg(n)}(\mathbb{F}_q), \mathbb{E}^{lg(n)}(\mathbb{F}_q), \mathbb{E}(\mathbb{F}_q), \mathbb{F}_q, \mathbb{E}(\mathbb{F}_q), \mathbb{F}_q)$

AccHiding = $(\mathbb{E}(\mathbb{F}_q), \mathbb{N}, \mathbb{F}_q, \mathbb{F}_q^d)$

Acc = $((\mathbb{E}(\mathbb{F}_q), \mathbb{N}, \mathbb{F}_q, \mathbb{F}_q, \mathbf{EvalProof}), \mathbf{AccHiding})$

CM: Pedersen Commitment

As a reference, we include the Pedersen Commitment algorithm we use:

Algorithm 9 CM.COMMIT

Inputs

$\mathbf{m} : \mathbb{F}^n$

The vectors we wish to commit to.

$\mathbf{G} : \mathbb{E}(\mathbb{F})^n$

The generators we use to create the commitment.

$\omega : \text{Option}(\mathbb{F}_q)$

Optional hiding factor for the commitment.

Output

$C : \mathbb{E}(\mathbb{F}_q)$

The pedersen commitment.

1: Output $C := \mathbf{m} \cdot \mathbf{G} + \omega S$.

```
pub fn commit(w: Option<&PallasScalar>, Gs: &[PallasAffine], ms: &[PallasScalar]) -> PallasPoint {
    assert!(
        Gs.len() == ms.len(),
        "Length did not match for pedersen commitment: {}, {}",
        Gs.len(),
        ms.len()
    );

    let acc = point_dot_affine(ms, Gs);
    if let Some(w) = w {
        S * w + acc
    } else {
        acc
    }
}
```