Veridise. Auditing Report

Hardening Blockchain Security with Formal Methods

FOR

Hydra-S2

Veridise Inc.

April 14, 2023

► **Prepared For:**

Sismo
https://www.sismo.io/

► **Prepared By:**

Jon Stephens
Hanzhi Liu

► **Contact Us:** contact@veridise.com

► **Version History:**

Apr. 11, 2023 Initial Draft

Contents

1 Executive Summary

From Mar. 21, 2023 to April 4, 2023, Sismo engaged Veridise to conduct a security review and formally verify the correctness of their Hydra-S2 Zero-Knowledge Circuits. These ZK circuits are used to validate private user information, including a user's digital identity proof, and computation for the Sismo protocol. Veridise conducted the assessment over 4 person-weeks, with 2 engineers reviewing code over 2 weeks on commit `0x2b79ab3`. The auditing strategy involved a tool-assisted analysis of the source code performed by Veridise engineers as well as extensive manual auditing. In parallel, the Veridise engineers also formally verified that the Hydra-S2 circuits adhere to the formal specifications shown in Section 5.

Code assessment. The Hydra-S2 circuits are part of the larger Sismo protocol that allows users to prove their digital identity and then use that digital identity to interact with other protocols without revealing it publicly. The Hydra-S2 circuits specifically allow users to privately prove their digital identity has been validated in addition to proving several application-specific properties. More specifically, the Hydra-S2 circuits allow users to prove that they know secret account information, that their secret account value is greater than or equal to some specified value, that their secret account value is equal to some specified value, that that the user knows how to construct some proof identifier and that the user knows how to compute some vault identifier. As the Hydra-S2 circuit has many functions, the Sismo developers allow the user to specify whether various actions should be enabled or disabled via flags or specific signal values.

Sismo provided the source code for the Hydra-S2 ZK circuits and the Sismo Commitment Mapper which was audited in parallel. A test suite accompanied the source code with tests written by developers. Additionally, the developers also provided documentation for the Hydra-S1 ZK circuits and the Hydra-S2 ZK circuits.

Summary of issues detected. The audit uncovered 1 issue that the Veridise auditors determined to be of moderate severity. This issue corresponds to the potential for information leakage due to a comparison between a private input and a public input.

Recommendations. After auditing the Hydra-S2 circuits, our auditors believe a likely source of errors may lie at the interface between Hydra-S2 and other applications. We believe that developers building on Hydra-S2 should pay special attention to this interface and should ensure that it is audited before deployment. In particular, we would like to point out several potential sources of error:

1. Those developing on Hydra-S2 should ensure they properly validate whether certain features are enabled or disabled. Most of the features in the circuit may be turned on or off in certain ways (by making a certain value zero or via a flag) and developers should ensure they properly identify when a feature is turned off. If developers are not careful, this could allow a protocol to believe a certain relationship has been verified.

2. Certain features should not be trusted in isolation, as in isolation there may not be sufficient validation to ensure that the user knows specific information. For example, the check guarded by `statementComparator` that ensures a user's `sourceValue` is equal to the `statementValue` should not be trusted unless `accountsTreeValue` is also non-zero.

Disclaimer. We hope that this report is informative but provide no warranty of any kind, explicit or implied. The contents of this report should not be construed as a complete guarantee that the system is secure in all dimensions. In no event shall Veridise or any of its employees be liable for any claim, damages or other liability, whether in an action of contract, tort or otherwise, arising from, out of or in connection with the results reported here.

2 Project Dashboard

Table 2.1: Application Summary.

Name	Version	Type	Platform
Hydra-S2	`0x2b79ab3`	Circom	N/A

Table 2.2: Engagement Summary.

Dates	Method	Consultants Engaged	Level of Effort
Mar. 21 - April 4, 2023	Manual & Tools	2	4 person-weeks

Table 2.3: Vulnerability Summary.

Name	Number	Resolved
Critical-Severity Issues	0	0
High-Severity Issues	0	0
Medium-Severity Issues	1	1
Low-Severity Issues	0	0
Warning-Severity Issues	0	0
Informational-Severity Issues	0	0
TOTAL	1	1

Table 2.4: Verification Summary.

Type	Number
Functional Correctness	5
Deterministic Circuit	1

Table 2.5: Vulnerability Category Breakdown.

Name	Number
Information Leakage	1

3 Audit Goals and Scope

3.1 Audit Goals

The engagement was scoped to provide a security assessment of the Hydra-S2 ZK Circuits. In our audit, we sought to answer the following questions:

- Can private information be leaked through public inputs or outputs?
- Are the signals in the circuits properly constrained?
- Does the circuit properly validate a user's EdDSA signature receipt?
- Can users determine which circuit checks were disabled by inspecting the public inputs or outputs?
- Can circuit functions properly be disabled using the various enable/disable flags?
- Does the circuit properly validate that a leaf is contained in the merkle tree with the given root?
- Where appropriate, do the developers validate that signals are within an appropriate range?
- Do all public inputs participate in at least one constraint?

3.2 Audit Methodology & Scope

Audit Methodology. To address the questions above, our audit involved a combination of manual inspection by human experts and automated program analysis & testing. In particular, we conducted our audit with the aid of the following techniques:

- *Static analysis.* To identify potential common vulnerabilities, we leveraged our custom Zero-Knowledge circuit static analysis tool. This tool is designed to find instances of common vulnerabilities in Zero-Knowledge circuits, such as unused public inputs and dataflow-constraint discrepancies.
- *Formal Verification.* To prove the correctness of the ZK circuits we used a combination of Coda, our formal verification project based on the Coq interactive theorem prover, and Picus, our automated verification tool. To do this, we formalized the intended behavior of the Circom templates and then proved the correctness of the implementation with respect to the formalized specifications.

Scope. The scope of this audit is limited to the `circuits` directory of the Hydra-S2 repository, which contains the source code of the Hydra-S2 ZK circuits. While other files were included in the source code, they were not in the scope of the audit. During the audit, the Veridise auditors referred to the excluded files but assumed that they have been implemented correctly.

Methodology. The Veridise auditors first inspected the provided tests and documentation to better understand the desired behavior of the provided source code at a more granular level. They then formalized the intended behavior of the Hydra-S2 circuits and formally verified

them with the help of Coda. In parallel, they performed a multi-week manual audit of the code assisted by our static analyzer.

3.3 Classification of Vulnerabilities

When Veridise auditors discover a possible security vulnerability, they must estimate its severity by weighing its potential impact against the likelihood that a problem will arise. Table 3.1 shows how our auditors weigh this information to estimate the severity of a given issue.

Table 3.1: Severity Breakdown.

	Somewhat Bad	Bad	Very Bad	Protocol Breaking
Not Likely	Info	Warning	Low	Medium
Likely	Warning	Low	Medium	High
Very Likely	Low	Medium	High	Critical

In this case, we judge the likelihood of a vulnerability as follows in Table 3.2:

Table 3.2: Likelihood Breakdown

Not Likely	A small set of users must make a specific mistake
Likely	Requires a complex series of steps by almost any user(s) - OR - Requires a small set of users to perform an action
Very Likely	Can be easily performed by almost anyone

In addition, we judge the impact of a vulnerability as follows in Table 3.3:

Table 3.3: Impact Breakdown

Somewhat Bad	Inconveniences a small number of users and can be fixed by the user
Bad	Affects a large number of people and can be fixed by the user - OR - Affects a very small number of people and requires aid to fix
Very Bad	Affects a large number of people and requires aid to fix - OR - Disrupts the intended behavior of the protocol for a small group of users through no fault of their own
Protocol Breaking	Disrupts the intended behavior of the protocol for a large group of users through no fault of their own

4 Vulnerability Report

In this section, we describe the vulnerabilities found during our audit. For each issue found, we log the type of the issue, its severity, location in the code base, and its current status (i.e., acknowleged, fixed, etc.). Table 4.1 summarizes the issues discovered:

Table 4.1: Summary of Discovered Vulnerabilities.

ID	Description	Severity	Status
V-SH2-VUL-001	Private Information Leakage	Medium	Acknowledged

4.1 Detailed Description of Issues

4.1.1 V-SH2-VUL-001: Private Information Leakage

Severity	Medium	**Address**	`0x2b79ab3`
Type	Information Leakage	**Status**	Acknowledged
File(s)	hydra-s2.circom		
Location(s)	hydraS2		

Zero-Knowledge circuits partition inputs into those that can be publicly revealed and those that should be kept private as they may contain secret information. During the process of evaluating a circuit, however, information about the private inputs may be leaked, allowing others to learn information about user secrets. This can occur in the `hydraS2` template as shown in Snippet 4.1. By setting the `statementComparator`, users are given the option of revealing their secret `sourceValue` as, in that case, it must be equal to the public `statementValue`. However even in the case where `statementComparator` is not set, information is leaked. This is because `sourceValue` is always checked to determine if it is greater than or equal to `statementValue`. Should this functionality be used, it is therefore the case that a lower-bound on the source value is leaked.

Impact Depending on how the circuit is used, this could allow a user to accidentally publish their `sourceValue` even in cases when they do not intend to. Based on the documentation, it appears that the leaked data would depend on the application using Hydra-S2, making it difficult to determine the potential impact this may have.

Recommendations Consider making the decision to enable the comparison between `sourceValue` and `statementValue` explicit similar to the equality check between `sourceValue` and `statementValue` .

Developer Response This comparison is a feature that allows users to prove that they meet some criteria that may be published later. If a user does not intend to use the lower-bound check, they can provide a `statementValue` of zero. In most cases where the feature is used, the `statementValue` will also likely be low, which would not reveal significant information about the `sourceValue`. There is the potential that this could be abused by requiring users reveal multiple proofs, but doing so would likely require a significant number of proofs to be provided.

```
template hydraS2(registryTreeHeight, accountsTreeHeight) {
  // Private inputs
  ...
  signal input sourceValue;

  // Public inputs
  ...
  signal input statementValue;
  signal input statementComparator;
  ...

  // Verify statement value validity
  // 0 => sourceValue can be higher than statementValue
  // 1 => sourceValue and statementValue should be equal
  // Prevent overflow of comparator range
  component sourceInRange = Num2Bits(252);
  sourceInRange.in <== sourceValue;
  component statementInRange = Num2Bits(252);
  statementInRange.in <== statementValue;
  // 0 <= statementValue <= sourceValue
  component leq = LessEqThan(252);
  leq.in[0] <== statementValue;
  leq.in[1] <== sourceValue;
  leq.out === 1;
  // If statementComparator == 1 then statementValue == sourceValue
  0 === (statementComparator-1)*statementComparator;
  sourceValue === sourceValue+((statementValue-sourceValue)*statementComparator);

  ...
}
```

Snippet 4.1: Snippet of the main Hydra-S2 template that leaks a lower bound on the `sourceValue`

5 Formal Verification

In this section, we describe the specifications that were used to formally verify the correctness of the ZK circuits. For each specification, we log its current status (i.e. verified, not verified). Table 5.1 summarizes the specifications and their verification status:

Table 5.1: Summary of the Formally Verified Properties.

ID	Description	Status
V-SH2-SPEC-001	PositionSwitcher Functional Correctness	Verified
V-SH2-SPEC-002	VerifyMerklePath Functional Correctness	Verified
V-SH2-SPEC-003	VerifyHydraCommitment Functional Correctness	Verified
V-SH2-SPEC-004	hydraS2 Functional Correctness	Verified
V-SH2-SPEC-005	EdDSAPoseidonVerifier Functional Correctness	Verified
V-SH2-SPEC-006	Poseidon is Deterministic	Verified

5.1 Detailed Description of Formal Verification Results

5.1.1 V-SH2-SPEC-001: PositionSwitcher Functional Correctness

Commit	`0x2b79ab3`	**Status**	Verified
Files	verify-merkle-path.circom		
Functions	PositionSwitcher		

Description The output $\{in[0], in[1]\}$ if s is 0 and $\{in[1], in[0]\}$ if s is 1.

Informal Specification

$$s \in \{0, 1\}$$

$$s = 0 \rightarrow out[0] = in[0] \land out[1] = in[1]$$

$$s = 1 \rightarrow out[0] = in[1] \land out[1] = in[0]$$

Formal Definition The following shows the formal definition for the `PositionSwitcher` template:

```
Definition cons (_in: (F^2)) (s: F) (out: F^2) :=
  s * (1 - s) = 0 /\
  out[0] = (_in[1] - _in[0])*s + _in[0] /\
  out[1] = (_in[0] - _in[1])*s + _in[1].
```

Formal Specification The following shows the formal specification for the `PositionSwitcher` template:

```
Definition spec (m: t) : Prop :=
  binary m.(s) /\
  (m.(s) = 0 -> m.(out)[0] = m.(_in)[0] /\ m.(out)[1] = m.(_in)[1]) /\
  (m.(s) = 1 -> m.(out)[0] = m.(_in)[1] /\ m.(out)[1] = m.(_in)[0]).
```

Proof The following shows the soundness proof for the `PositionSwitcher` template:

```
Theorem soundness:
  forall (c: t), spec c.
Proof.
  unwrap_C.
  intros c.
  destruct c as [_in s out _cons1].
  unfold spec, cons in *. simpl.
  split;intros;intuit.
  - unfold binary. destruct (dec (s = 0));try fqsatz;auto.
    destruct (dec (s = 1));try fqsatz;auto.
  - subst. fqsatz.
  - subst. fqsatz.
  - subst. fqsatz.
```

```
14 |   - subst. fqsatz.
15 | Qed.
```

5.1.2 V-SH2-SPEC-002: VerifyMerklePath Functional Correctness

Commit	0x2b79ab3	Status	Verified
Files	verify-merkle-path.circom		
Functions	VerifyMerklePath		

Description If the circuit is enabled, the circuit will check that the given root matches the computed merkle tree root.

Informal Specification

$$\forall_{0 \le i < \text{levels}} i.\ \text{pathIndices}[i] \in \{0, 1\}$$

$$\text{computedPath}(i) := \begin{cases} \text{poseidon}_2(\text{computedPath}(i-1), \text{pathElements}[i]) & \text{if pathIndices}[i] = 0 \\ \text{poseidon}_2(\text{pathElements}[i], \text{computedPath}(i-1)) & \text{if pathIndices}[i] = 1 \\ \text{leaf} & \text{if } i = -1 \end{cases}$$

$$\text{enabled} \rightarrow \text{root} = \text{computedPath(levels-1)}$$

Formal Definition The following shows the formal definition for the `VerifyMerklePath` template:

```
Definition cons (leaf: F) (root: F) (enabled: F) (pathElements: F^levels) (
    pathIndices: F^levels) :=
  exists (selectors: PositionSwitcher.t^levels) (hashers: (@Poseidon.t 2)^levels) (
    computedPath: F^levels),
  let _C :=
    (D.iter (fun i _C =>
    _C /\
    selectors[i].(PositionSwitcher._in)[0] = (if i =? 0 then leaf else computedPath[i
     - 1]) /\
    selectors[i].(PositionSwitcher._in)[1] = pathElements[i] /\
    selectors[i].(PositionSwitcher.s) = pathIndices[i] /\
    hashers[i].(Poseidon.inputs)[0] = selectors[i].(PositionSwitcher.out)[0] /\
    hashers[i].(Poseidon.inputs)[1] = selectors[i].(PositionSwitcher.out)[1] /\
    computedPath[i] = hashers[i].(Poseidon.out))levels True)
  in _C /\
  (root - computedPath[levels - 1])*enabled = 0.
```

Formal Specification The following shows the formal specification for the `VerifyMerklePath` template:

```
Definition spec (c: t) : Prop :=
  (forall i, 0 <= i < levels -> binary (c.(pathIndices)[i])) /\
  (c.(enabled) <> 0 -> c.(root) = fold_left (fun (y:F) (x:(F*F)) => if dec (fst x =
    0) then (poseidon_2 y (snd x)) else (poseidon_2 (snd x) y))
                       (combine ('(c.(pathIndices))) ('(c.(pathElements)))) c.(leaf))
    .
```

Proof The following shows the soundness proof for the `VerifyMerklePath` template:

```
Lemma fold_left_firstn_S:
  forall (l: list (F*F))(i: nat)(b: F)f,
  i < length l ->
  fold_left f  (l [:S i]) b =
  f (fold_left f (l [:i]) b) (l ! i).
Proof.
  intros.
  assert(l [:S i] = l [:i] ++ ((l ! i)::nil)).
  { erewrite firstn_S;try lia. unfold_default. auto. }
  rewrite H0.
  apply fold_left_app.
Qed.

Lemma combine_fst_n: forall n j (l1 l2: F^n),
  j < n ->
  j < n ->
  fst (combine (' l1) (' l2) ! j) = l1 [j].
Proof.
  intros. pose proof (length_to_list l1). pose proof (length_to_list l2).
  unfold_default. simpl. rewrite combine_nth;simpl;auto.
  rewrite nth_Default_nth_default. rewrite <- nth_default_to_list. unfold_default.
    auto.
  rewrite H1, H2;auto.
Qed.

Lemma combine_snd_n: forall n j (l1 l2: F^n),
  j < n ->
  j < n ->
  snd (combine (' l1) (' l2) ! j) = l2 [j].
Proof.
  intros. pose proof (length_to_list l1). pose proof (length_to_list l2).
  unfold_default. simpl. rewrite combine_nth;simpl;auto.
  rewrite nth_Default_nth_default. rewrite <- nth_default_to_list. unfold_default.
    auto.
  rewrite H1, H2;auto.
Qed.

(* VerifyMerklePathProof is sound *)
Theorem soundness:
  forall (c: t), spec c.
Proof.
  unwrap_C.
  intros c.
  destruct c as [leaf root enabled pathElements pathIndices _cons].
  unfold spec, cons in *. simpl.
  destruct _cons as [switchs _cons]. destruct _cons as [poseidons _cons]. destruct
    _cons as [hashes _cons].
  destruct _cons as [_cons1 _cons2].
  rem_iter. subst. rem_iter.
  pose (Inv1 := fun (i: nat) (_cons: Prop) => _cons ->
                (forall j, j < i -> binary ((pathIndices)[j]))).
  assert (HInv1: Inv1 levels (D.iter f levels True)).
```



```
  { apply D.iter_inv; unfold Inv1;intros;try lia.
    subst. destruct H1. destruct (dec (j0 = j));intuit.
    + subst.
      pose proof (PositionSwitcher.soundness (switchs [j])). unfold PositionSwitcher.
    spec in H.
      intuit. unfold binary in *. rewrite H5 in *. auto.
    + apply H8;auto. lia. }
  apply HInv1 in _cons1 as inv1.
  split;intros. apply inv1;lia.
  pose (Inv2 := fun (i: nat) (_cons: Prop) => _cons ->
               (hashes [i - 1] = (fold_left
               (fun (y : F) (x : F * F) => if dec (fst x = 0) then poseidon_2 y (snd
   x) else poseidon_2 (snd x) y)
               (firstn i (combine (' pathIndices) (' pathElements)))
               (leaf)))).
  assert (HInv2: Inv2 levels (D.iter f levels True)).
  { apply D.iter_inv; unfold Inv2;intros;try lia.
    + simpl. auto. skip.
    + subst. destruct H2.
      do 5 destruct H3 as [? H3].
      pose proof (PositionSwitcher.soundness (switchs [j])). unfold PositionSwitcher.
    spec in H9.
      erewrite (fold_left_firstn_S (combine (' pathIndices) (' pathElements)));simpl.
      2:{ pose_lengths. rewrite combine_length. rewrite _Hlen4, _Hlen3. lia. }
      assert(FST: (fst (combine (' pathIndices) (' pathElements) ! j) = pathIndices [
    j])).
      { rewrite combine_fst_n;auto. }
      assert(SND: (snd (combine (' pathIndices) (' pathElements) ! j) = pathElements
    [j])).
      { rewrite combine_snd_n;auto. }
      rewrite FST, SND in *. destruct H9, H10. pose proof (H0 H2) as HASHJ.
      replace (j - 0)%nat with j by lia.
      destruct (dec (pathIndices [j] = 0)).
      ++ rewrite e in *. pose proof (H10 H6). inversion H12.
         rewrite H13 in H7;try lia. rewrite H14 in H8;try lia.
         rewrite HASHJ in H4. rewrite H4, H5 in *.
         rewrite H3. apply Poseidon.PoseidonHypo.poseidon_2_spec;auto.
         rewrite H7. destruct j;try easy.
      ++ pose proof (inv1 j). destruct H12;try lia;try easy. rewrite H12 in *.
         pose proof (H11 H6). destruct H13.
         rewrite H13 in *;try lia. rewrite H14 in *;try lia.
         rewrite H4, H5 in *.
         rewrite HASHJ in H8. rewrite H3. apply Poseidon.PoseidonHypo.poseidon_2_spec
    ;auto.
          rewrite H8. destruct j;try easy.
  }
  apply HInv2 in _cons1 as inv2.
  assert (root = hashes [levels - 1]). fqsatz. subst.
  rewrite inv2. rewrite combine_firstn. pose_lengths.
  assert((' pathElements [:levels]) = (' pathElements)).
  { rewrite <- _Hlen1 at 1. apply ListUtil.List.firstn_all. }
  rewrite <- _Hlen0 at 1. rewrite ListUtil.List.firstn_all. rewrite H0. auto.
Qed.
```

```
97
98  Theorem circuit_disabled (leaf: F) (enabled: F) (pathElements: F^levels) (pathIndices
        : F^levels)
99  (selectors: PositionSwitcher.t^levels) (hashers: (@Poseidon.t 2)^levels) (
        computedPath: F^levels):
100 let _C :=
101   (D.iter (fun i _C =>
102   _C /\
103   selectors[i].(PositionSwitcher._in)[0] = (if i =? 0 then leaf else computedPath[i -
        1]) /\
104   selectors[i].(PositionSwitcher._in)[1] = pathElements[i] /\
105   selectors[i].(PositionSwitcher.s) = pathIndices[i] /\
106   hashers[i].(Poseidon.inputs)[0] = selectors[i].(PositionSwitcher.out)[0] /\
107   hashers[i].(Poseidon.inputs)[1] = selectors[i].(PositionSwitcher.out)[1] /\
108   computedPath[i] = hashers[i].(Poseidon.out))levels True)
109 in _C ->
110 forall root,
111 enabled = 0 ->
112 (root - computedPath[levels - 1])*enabled = 0.
113 Proof.
114   intros.
115   subst. rewrite Fmul_0_r. auto.
116 Qed.
```

5.1.3 V-SH2-SPEC-003: VerifyHydraCommitment Functional Correctness

Commit	`0x2b79ab3`	**Status**	Verified
Files	verify-hydra-commitment.circom		
Functions	VerifyHydraCommitment		

Description If the circuit is enabled, it will verify that the private key associated with the given public key was used to sign a message corresponding to the hashes and values of `vaultSecret`, `accountSecret` and `address` to produce the given receipt.

Informal Specification

$$\text{message} := \text{poseidon}_2(\text{address}, \text{poseidon}_2(\text{vaultSecret}, \text{accountSecret}))$$

$$\text{enabled} \neq 0 \rightarrow \text{eddsa\_poseidon}(\text{commitmentMapperPubKey}[0], \text{commitmentMapperPubKey}[1], \text{commitmentReceipt}[0], \text{commitmentReceipt}[1], \text{commitmentReceipt}[2], \text{message})$$

Formal Definition The following shows the formal definition for the `VerifyHydraCommitment` template:

```
Definition cons (address: F)(accountSecret: F)(vaultSecret: F)(enabled: F)
                (commitmentMapperPubKey: F^2)(commitmentReceipt: F^3): Prop :=
  exists (commitment: (@Poseidon.t 2)) (message: (@Poseidon.t 2)) (eddsa:
    EdDSAPoseidonVerifier.t),
    commitment.(Poseidon.inputs)[0] = vaultSecret /\
    commitment.(Poseidon.inputs)[1] = accountSecret /\
    message.(Poseidon.inputs)[0] = address /\
    message.(Poseidon.inputs)[1] = commitment.(Poseidon.out) /\
    eddsa.(EdDSAPoseidonVerifier.enabled) = enabled /\
    eddsa.(EdDSAPoseidonVerifier.Ax) = commitmentMapperPubKey[0] /\
    eddsa.(EdDSAPoseidonVerifier.Ay) = commitmentMapperPubKey[1] /\
    eddsa.(EdDSAPoseidonVerifier.R8x) = commitmentReceipt[0] /\
    eddsa.(EdDSAPoseidonVerifier.R8y) = commitmentReceipt[1] /\
    eddsa.(EdDSAPoseidonVerifier.S) = commitmentReceipt[2] /\
    eddsa.(EdDSAPoseidonVerifier.M) = message.(Poseidon.out).
```

Formal Specification The following shows the formal specification for the `VerifyHydraCommitment` template:

```
Definition spec (c: t): Prop :=
  c.(enabled) <> 0 ->
  let message := poseidon_2 c.(address) (poseidon_2 c.(vaultSecret) c.(accountSecret)
    ) in
  eddsa_poseidon (c.(commitmentMapperPubKey)[0]) (c.(commitmentMapperPubKey)[1]) (c.(
    commitmentReceipt)[2]) (c.(commitmentReceipt)[0]) (c.(commitmentReceipt)[1])
    message.
```

Proof The following shows the soundness proof for the `VerifyHydraCommitment` template:

```
Theorem soundness: forall (c: t), spec c.
Proof.
  intros. unfold spec. intuition.
  destruct c. simpl in *. intuition.
  destruct _cons0 as [commitment _cons].
  destruct _cons as [message _cons].
  destruct _cons as [eddsa _cons].
  intuition.
  pose proof (Poseidon.PoseidonHypo.poseidon_2_spec commitment) as commitment_spec.
  pose proof (Poseidon.PoseidonHypo.poseidon_2_spec message) as message_spec.
  pose proof (CircomLib.EdDSA.EdDSAPoseidonVerifier.EdDSAPoseidonVerifierProof.
    EdDSAPoseidonVerifier_spec eddsa) as eddsa_spec.
  intuit. subst.
  apply eddsa_spec in H.
  rewrite H5, H6, H7, H8, H9, H11 in H. unfold eddsa_poseidon. unfold poseidon_2.
  rewrite <- message_spec;auto.
  rewrite <- commitment_spec;auto.
Qed.

Definition tt: Prop := True.

Theorem circuit_disabled: forall (c:t), c.(enabled) = 0 -> tt.
Proof.
  intros.
  destruct c. simpl in *. intuition.
  destruct _cons0 as [commitment _cons].
  destruct _cons as [message _cons].
  destruct _cons as [eddsa _cons].
  intuit.
  destruct eddsa as [enabled Ax Ay R8x R8y S M cons]. simpl in *.
  pose proof (CircomLib.EdDSA.EdDSAPoseidonVerifier.EdDSAPoseidonVerifierProof.
    EdDSAPoseidonVerifier_spec_disabled
  Ax Ay R8x R8y S M) as eddsa_spec_disabled.
  destruct cons. do 2 destruct H10. intuit. subst.
  specialize (eddsa_spec_disabled x x0 x1).
  intuit.
  destruct CircomLib.EdDSA.EdDSAPoseidonVerifier.BabyDbl.
  destruct CircomLib.EdDSA.EdDSAPoseidonVerifier.BabyDbl.
  destruct CircomLib.EdDSA.EdDSAPoseidonVerifier.BabyDbl.
  destruct CircomLib.EdDSA.EdDSAPoseidonVerifier.edwards_mult.
  destruct CircomLib.EdDSA.EdDSAPoseidonVerifier.edwards_add.
  destruct CircomLib.EdDSA.EdDSAPoseidonVerifier.edwards_mult.
  intuit.
  (* signals are unconstrained *)
  assert(CircomLib.EdDSA.EdDSAPoseidonVerifier.EdDSAPoseidonVerifierProof.
    unconstrained
  f9 f7). auto.
  assert(CircomLib.EdDSA.EdDSAPoseidonVerifier.EdDSAPoseidonVerifierProof.
    unconstrained
  f10 f8). auto.
  unfold tt. auto.
Qed.
```


5.1.4 V-SH2-SPEC-004: hydraS2 Functional Correctness

Commit	`0x2b79ab3`	**Status**	Verified
Files	hydra-s2.circom		
Functions	hydraS2		

Description The `hydraS2` circuit will:

1. Verify the EdDSA poseidon signature corresponding to the source if enabled
2. Verify the EdDSA poseidon signature corresponding to the destination if enabled
3. Verify the account identifier is in the merkle tree with the specified `accountsTreeRoot` if enabled
4. Verify the account tree is in the merkle tree with the specified `registryTreeRoot` if enabled
5. Verify the `statementValue` is less than or equal to the `sourceValue`
6. Verify the `statementValue` is equal to the `sourceValue` if enabled
7. Verify the `proofIdentifier` hash combines the values and hashes of `sourceSecret`, 1, `sourceSecretHash` and `requestIdentifier` if enabled
8. Veirfy the `vaultIdentifier` hash equals the poseidon hash of the `vaultSecret` and `vaultNamespace` if enabled

Informal Specification

$$\text{VerifyHydraCommitment}(\text{sourceIdentifier}, \text{vaultSecret}, \text{sourceSecret}, \text{sourceVerificationEnabled}, \text{commitmentMapperPubKey}, \text{sourceCommitmentReceipt})$$

$$\text{VerifyHydraCommitment}(\text{destinationIdentifier}, \text{vaultSecret}, \text{destinationSecret}, \text{destinationVerificationEnabled}, \text{commitmentMapperPubKey}, \text{destinationCommitmentReceipt})$$

$$\text{accountLeaf} := \text{poseidon}_2(\text{sourceIdentifier}, \text{sourceValue})$$

$$\text{VerifyMerklePath}(\text{accountLeaf}, \text{accountsTreeRoot}, \text{accountsTreeValue} \neq 0, \text{accountMerklePathElements}, \text{accountMerklePathIndices})$$

$$\text{registryLeaf} := \text{poseidon}_2(\text{sourceIdentifier}, \text{sourceValue})$$

$$\text{VerifyMerklePath}(\text{registryLeaf}, \text{registryTreeRoot}, \text{accountsTreeValue} \neq 0, \text{registryMerklePathElements}, \text{registryMerklePathIndices})$$

$$\text{sourceValue} < 2^{252} \land \text{statementValue} < 2^{252} \land 0 \leq \text{statementValue} \leq \text{sourceValue}$$

$$\text{statementComparator} = 1 \rightarrow \text{statementValue} = \text{sourceValue}$$

$$\text{requestIdentifier} \neq 0 \rightarrow \text{proofIdentifier} = \text{poseidon}_2(\text{sourceSecretHash}, \text{requestIdentifier})$$

$$\text{vaultNamespace} \neq 0 \rightarrow \text{vaultIdentifier} = \text{poseidon}_2(\text{vaultSecret}, \text{vaultNamespace})$$

Formal Definition The following shows the formal definition for the `hydraS2` template:

```
Definition cons (sourceIdentifier: F) (sourceSecret: F) (sourceValue: F) (vaultSecret: F) (sourceCommitmentReceipt: F^3)
                (destinationIdentifier: F) (destinationSecret: F) (destinationCommitmentReceipt: F^3)
                (accountMerklePathElements: F^accountsTreeHeight) (accountMerklePathIndices: F^accountsTreeHeight) (accountsTreeRoot: F)
                (registryMerklePathElements: F^registryTreeHeight) (registryMerklePathIndices: F^registryTreeHeight) (registryTreeRoot: F)
                (extraData: F) (commitmentMapperPubKey: F^2)
                (requestIdentifier: F) (proofIdentifier: F)
                (statementValue: F) (statementComparator: F) (accountsTreeValue: F)
                (vaultIdentifier: F) (vaultNamespace: F)
                (sourceVerificationEnabled: F) (destinationVerificationEnabled: F) :
    Prop :=
exists (sourceCommitmentVerification: VerifyHydraCommitment.t) (destinationCommitmentVerification: VerifyHydraCommitment.t)
       (accountsTreeValueIsZero: IsZero.t) (accountLeafConstructor: @Poseidon.t 2) (accountsTreesPathVerifier: @VerifyMerklePath.t accountsTreeHeight)
       (registryLeafConstructor: @Poseidon.t 2) (registryTreePathVerifier: @VerifyMerklePath.t registryTreeHeight)
       (sourceInRange: @Num2Bits.t 252) (statementInRange: @Num2Bits.t 252) (leq: @LessEqThan.t 252)
       (sourceSecretHash: F) (sourceSecretHasher: @Poseidon.t 2) (requestIdentifierIsZero: IsZero.t) (proofIdentifierHasher: @Poseidon.t 2)
       (vaultNamespaceIsZero: IsZero.t) (vaultIdentifierHasher: @Poseidon.t 2),
  (* Verify the source account went through the Hydra Delegated Proof of Ownership
     That means the user own the source address *)
  sourceCommitmentVerification.(VerifyHydraCommitment.address) = sourceIdentifier /\
  sourceCommitmentVerification.(VerifyHydraCommitment.vaultSecret) = vaultSecret /\
  sourceCommitmentVerification.(VerifyHydraCommitment.accountSecret) = sourceSecret /\
  sourceCommitmentVerification.(VerifyHydraCommitment.enabled) = sourceVerificationEnabled /\
  sourceCommitmentVerification.(VerifyHydraCommitment.commitmentMapperPubKey) = commitmentMapperPubKey /\
  sourceCommitmentVerification.(VerifyHydraCommitment.commitmentReceipt) = sourceCommitmentReceipt /\
  (* Verify the destination account went through the Hydra Delegated Proof of Ownership
     That means the user own the destination address *)
  destinationCommitmentVerification.(VerifyHydraCommitment.address) = destinationIdentifier /\
  destinationCommitmentVerification.(VerifyHydraCommitment.vaultSecret) = vaultSecret /\
  destinationCommitmentVerification.(VerifyHydraCommitment.accountSecret) = destinationSecret /\
  destinationCommitmentVerification.(VerifyHydraCommitment.enabled) = destinationVerificationEnabled /\
  destinationCommitmentVerification.(VerifyHydraCommitment.commitmentMapperPubKey) = commitmentMapperPubKey /\
  destinationCommitmentVerification.(VerifyHydraCommitment.commitmentReceipt) = destinationCommitmentReceipt /\
  accountsTreeValueIsZero.(IsZero._in) = accountsTreeValue /\
```



```
  (* Recreating the leaf *)
  accountLeafConstructor.(Poseidon.inputs)[0] = sourceIdentifier /\
  accountLeafConstructor.(Poseidon.inputs)[1] = sourceValue /\
  accountsTreesPathVerifier.(VerifyMerklePath.leaf) = accountLeafConstructor.(
    Poseidon.out) /\
  accountsTreesPathVerifier.(VerifyMerklePath.root) = accountsTreeRoot /\
  accountsTreesPathVerifier.(VerifyMerklePath.enabled) = (1 - accountsTreeValueIsZero
    .(IsZero.out)) /\
  accountsTreesPathVerifier.(VerifyMerklePath.pathElements) =
    accountMerklePathElements /\
  accountsTreesPathVerifier.(VerifyMerklePath.pathIndices) = accountMerklePathIndices
     /\
  (* Recreating the leaf *)
  registryLeafConstructor.(Poseidon.inputs)[0] = accountsTreeRoot /\
  registryLeafConstructor.(Poseidon.inputs)[1] = accountsTreeValue /\
  registryTreePathVerifier.(VerifyMerklePath.leaf) = registryLeafConstructor.(
    Poseidon.out) /\
  registryTreePathVerifier.(VerifyMerklePath.root) = registryTreeRoot /\
  registryTreePathVerifier.(VerifyMerklePath.enabled) = (1 - accountsTreeValueIsZero
    .(IsZero.out)) /\
  registryTreePathVerifier.(VerifyMerklePath.pathElements) =
    registryMerklePathElements /\
  registryTreePathVerifier.(VerifyMerklePath.pathIndices) = registryMerklePathIndices
     /\
  (* Verify statement value validity *)
  sourceInRange.(Num2Bits._in) = sourceValue /\
  statementInRange.(Num2Bits._in) = statementValue /\
  leq.(LessEqThan._in)[0] = statementValue /\
  leq.(LessEqThan._in)[1] = sourceValue /\
  leq.(LessEqThan.out) = 1 /\
  0 = (statementComparator - 1) * statementComparator /\
  sourceValue = sourceValue+((statementValue-sourceValue)*statementComparator) /\
  (* Verify the proofIdentifier is valid
     compute the sourceSecretHash using the hash of the sourceSecret *)
  sourceSecretHasher.(Poseidon.inputs)[0] = sourceSecret /\
  sourceSecretHasher.(Poseidon.inputs)[1] = 1 /\
  sourceSecretHash = sourceSecretHasher.(Poseidon.out) /\
  (* Check the proofIdentifier is valid only if requestIdentifier is not 0 *)
  requestIdentifierIsZero.(IsZero._in) = requestIdentifier /\
  proofIdentifierHasher.(Poseidon.inputs)[0] = sourceSecretHash /\
  proofIdentifierHasher.(Poseidon.inputs)[1] = requestIdentifier /\
  (proofIdentifierHasher.(Poseidon.out) - proofIdentifier) * (1 -
    requestIdentifierIsZero.(IsZero.out)) = 0 /\
  (* Compute the vaultIdentifier *)
  vaultNamespaceIsZero.(IsZero._in) = vaultNamespace /\
  vaultIdentifierHasher.(Poseidon.inputs)[0] = vaultSecret /\
  vaultIdentifierHasher.(Poseidon.inputs)[1] = vaultNamespace /\
  (vaultIdentifierHasher.(Poseidon.out) - vaultIdentifier) * (1 -
    vaultNamespaceIsZero.(IsZero.out)) = 0.
```

Formal Specification The following shows the formal specification for the `hydraS2` template:

```
Definition spec (c: t): Prop :=
```

```
  (* (1) *)
  ( c.(sourceVerificationEnabled) <> 0 ->
    let sourceSecretHash := poseidon_2 c.(sourceIdentifier) (poseidon_2 c.(
    vaultSecret) c.(sourceSecret)) in
    eddsa_poseidon (c.(commitmentMapperPubKey)[0]) (c.(commitmentMapperPubKey)[1]) (c
    .(sourceCommitmentReceipt)[2]) (c.(sourceCommitmentReceipt)[0]) (c.(
    sourceCommitmentReceipt)[1]) sourceSecretHash) /\
  (* (2) *)
  ( c.(destinationVerificationEnabled) <> 0 ->
    let destinationSecretHash := poseidon_2 c.(destinationIdentifier) (poseidon_2 c.(
    vaultSecret) c.(destinationSecret)) in
    eddsa_poseidon (c.(commitmentMapperPubKey)[0]) (c.(commitmentMapperPubKey)[1]) (c
    .(destinationCommitmentReceipt)[2]) (c.(destinationCommitmentReceipt)[0]) (c.(
    destinationCommitmentReceipt)[1]) destinationSecretHash) /\
  (* (3) *)
  ( c.(accountsTreeValue) <> 0 ->
    let leaf := poseidon_2 c.(sourceIdentifier) c.(sourceValue) in
    c.(accountsTreeRoot) = fold_left (fun (y:F) (x:(F*F)) => if dec (fst x = 0) then
    (poseidon_2 y (snd x)) else (poseidon_2 (snd x) y)) (combine ('(c.(
    accountMerklePathIndices))) ('(c.(accountMerklePathElements)))) leaf) /\
  (* (4) *)
  ( c.(accountsTreeValue) <> 0 ->
    let leaf := poseidon_2 c.(accountsTreeRoot) c.(accountsTreeValue) in
    c.(registryTreeRoot) = fold_left (fun (y:F) (x:(F*F)) => if dec (fst x = 0) then
    (poseidon_2 y (snd x)) else (poseidon_2 (snd x) y)) (combine ('(c.(
    registryMerklePathIndices))) ('(c.(registryMerklePathElements)))) leaf) /\
  (* (5) *)
  ( c.(sourceValue) | (252) /\ c.(statementValue) | (252) /\ c.(statementValue) <=q c
    .(sourceValue)) /\
  (* (6) *)
  ( c.(statementComparator) = 1 -> c.(statementValue) = c.(sourceValue)) /\
  (* (7) *)
  ( c.(requestIdentifier) <> 0 -> c.(proofIdentifier) = poseidon_2 (poseidon_2 c.(
    sourceSecret) 1%F) c.(requestIdentifier)) /\
  (* (8) *)
  ( c.(vaultNamespace) <> 0 -> c.(vaultIdentifier) = poseidon_2 c.(vaultSecret) c.(
    vaultNamespace))
  .
```

Proof The following shows the soundness proof for the `hydraS2` template:

```
Ltac destruct_cons _cons0 :=
  destruct _cons0 as [sourceCommitmentVerification _cons];
  destruct _cons as [destinationCommitmentVerification _cons];
  destruct _cons as [accountsTreeValueIsZero _cons];
  destruct _cons as [accountLeafConstructor _cons];
  destruct _cons as [accountsTreesPathVerifier _cons];
  destruct _cons as [registryLeafConstructor _cons];
  destruct _cons as [registryTreesPathVerifier _cons];
  destruct _cons as [sourceInRange _cons];
  destruct _cons as [statementInRange _cons];
  destruct _cons as [leq _cons];
  destruct _cons as [sourceSecretHash _cons];
```



```
13    destruct _cons as [sourceSecretHasher _cons];
14    destruct _cons as [requestIdentifierIsZero _cons];
15    destruct _cons as [proofIdentifierHasher _cons];
16    destruct _cons as [vaultNamespaceIsZero _cons];
17    destruct _cons as [vaultIdentifierHasher _cons];
18    simpl in *;intuit.
19
20  Lemma F_0_1_ff: 1%F <> @F.zero q.
21  Proof.
22  unwrap_C.
23  intro. pose proof @F.to_Z_0 q. rewrite <- H in H0. simpl in *. rewrite Zmod_1_l in H0
        ;try lia.
24  Qed.
25
26  Lemma F_sub_eq: forall (a b: F), a - b = 0 -> a = b.
27  Proof.
28    intros.
29    pose proof (F.ring_theory q). destruct H0.
30    apply Crypto.Algebra.Ring.sub_zero_iff;auto.
31  Qed.
32
33  Hypothesis CPLen: (C.k > 252)%Z.
34
35  (* HydraS2 is sound *)
36  Theorem soundness: forall (c: t), spec c.
37  Proof.
38    intros.
39    destruct c. simpl in *.
40    unfold spec. intuition;simpl.
41    - destruct_cons _cons0. subst.
42      pose proof (VerifyHydraCommitment.soundness sourceCommitmentVerification) as
        sourceCommitmentVerification_spec.
43      unfold VerifyHydraCommitment.spec in sourceCommitmentVerification_spec.
44      apply sourceCommitmentVerification_spec. auto.
45    - destruct_cons _cons0. subst.
46      pose proof (VerifyHydraCommitment.soundness destinationCommitmentVerification) as
         destinationCommitmentVerification_spec.
47      unfold VerifyHydraCommitment.spec in destinationCommitmentVerification_spec.
48      rewrite <- H10, <- H7.
49      apply destinationCommitmentVerification_spec. auto.
50    - destruct_cons _cons0. subst.
51      pose proof (VerifyMerklePath.soundness accountsTreesPathVerifier) as
        accountsTreesPathVerifier_spec.
52      unfold VerifyMerklePath.spec in accountsTreesPathVerifier_spec. intuit.
53      erewrite <- (Poseidon.PoseidonHypo.poseidon_2_spec);eauto.
54      rewrite H1;auto. rewrite H15;auto.
55      pose proof (IsZero.soundness accountsTreeValueIsZero) as
        accountsTreeValueIsZero_spec.
56      unfold IsZero.spec in accountsTreeValueIsZero_spec.
57      destruct (dec (IsZero._in accountsTreeValueIsZero = 0));try easy.
58      rewrite accountsTreeValueIsZero_spec in H17. rewrite H17. intro.
59      replace (1-0)%F with (@F.one q) in H2.
60      apply F_0_1_ff in H2;easy.
```

```
61      rewrite Fsub_0_r;auto.
62    - destruct_cons _cons0. subst.
63      pose proof (VerifyMerklePath.soundness registryTreesPathVerifier) as
        registryTreesPathVerifier_spec.
64      unfold VerifyMerklePath.spec in registryTreesPathVerifier_spec. intuit.
65      erewrite <- (Poseidon.PoseidonHypo.poseidon_2_spec);eauto.
66      rewrite H1;auto. rewrite H22;auto.
67      pose proof (IsZero.soundness accountsTreeValueIsZero) as
        accountsTreeValueIsZero_spec.
68      unfold IsZero.spec in accountsTreeValueIsZero_spec.
69      destruct (dec (IsZero._in accountsTreeValueIsZero = 0));try easy.
70      rewrite accountsTreeValueIsZero_spec in H24. rewrite H24. intro.
71      replace (1-0)%F with (@F.one q) in H2.
72      apply F_0_1_ff in H2;easy.
73      rewrite Fsub_0_r;auto.
74    - destruct_cons _cons0. subst.
75      pose proof (Num2Bits.range_check sourceInRange).
76      rewrite H26 in *. rewrite H;try lia.
77    - destruct_cons _cons0. subst.
78      pose proof (Num2Bits.range_check statementInRange).
79      rewrite H;try lia.
80    - destruct_cons _cons0. subst.
81      pose proof (LessEqThan.soundness leq). intuition.
82      destruct H;try lia;auto.
83      pose proof (Num2Bits.range_check statementInRange).
84      rewrite H28 in *. rewrite H;try lia.
85      pose proof (Num2Bits.range_check sourceInRange).
86      rewrite H26,H29 in *. rewrite H;try lia.
87      destruct (dec (LessEqThan.out leq = 1));try easy.
88      rewrite <- H28. rewrite H29 in H0. auto.
89    - destruct_cons _cons0. subst.
90      rewrite Fmul_1_r in H33.
91      assert (inputs accountLeafConstructor [1] +
92      (Num2Bits._in statementInRange - inputs accountLeafConstructor [1]) = Num2Bits.
        _in statementInRange).
93      { pose proof F.ring_theory q. destruct H. rewrite Rsub_def. rewrite Radd_assoc.
94        rewrite Radd_comm. rewrite Radd_assoc.
95        specialize (Ropp_def (inputs accountLeafConstructor [1])).
96        erewrite Radd_comm in Ropp_def. rewrite Ropp_def.
97        rewrite Fadd_0_l. auto. }
98      rewrite H in *. auto.
99    - destruct_cons _cons0. subst.
100     pose proof (Poseidon.PoseidonHypo.poseidon_2_spec proofIdentifierHasher).
101     pose proof (IsZero.soundness requestIdentifierIsZero) as
        requestIdentifierIsZero_spec.
102     unfold IsZero.spec in requestIdentifierIsZero_spec.
103     destruct (dec (IsZero._in requestIdentifierIsZero = 0));try easy.
104     rewrite requestIdentifierIsZero_spec in *. rewrite Fsub_0_r,Fmul_1_r in *.
105     assert (out proofIdentifierHasher = proofIdentifier0). apply F_sub_eq;auto.
106     erewrite <- H1, H0;eauto.
107     rewrite H38.
108     pose proof (Poseidon.PoseidonHypo.poseidon_2_spec sourceSecretHasher).
109     erewrite <- H2;eauto.
```



```
  - destruct_cons _cons0. subst.
    pose proof (Poseidon.PoseidonHypo.poseidon_2_spec vaultIdentifierHasher).
    pose proof (IsZero.soundness vaultNamespaceIsZero) as vaultNamespaceIsZero_spec.
    unfold IsZero.spec in vaultNamespaceIsZero_spec.
    destruct (dec (IsZero._in vaultNamespaceIsZero = 0));try easy.
    rewrite vaultNamespaceIsZero_spec in *. rewrite Fsub_0_r,Fmul_1_r in *.
    assert (out vaultIdentifierHasher = vaultIdentifier0). apply F_sub_eq;auto.
    erewrite <- H1, H0;eauto.
Qed.
```

5.1.5 V-SH2-SPEC-005: EdDSAPoseidonVerifier Functional Correctness

Commit	`0x2b79ab3`	**Status**	Verified
Files	eddsaposeidon.circom		
Functions	EdDSAPoseidonVerifier		

Description The template verifies that the account with the associated public key signed the given message using the EdDSA poseidon signature scheme to produce the given receipt.

Formal Definition The following shows the formal definition for the `EdDSAPoseidonVerifier` template:

```
Definition eddsa_poseidon Ax Ay S R8x R8y M :=
  let hash := poseidons_5 R8x R8y Ax Ay M in
  let '(dbl1_xout, dbl1_yout)  := BabyDbl Ax Ay in
  let '(dbl2_xout, dbl2_yout)  := BabyDbl dbl1_xout dbl1_yout in
  let '(dbl3_xout, dbl3_yout)  := BabyDbl dbl2_xout dbl2_yout in
  let '(right2_x, right2_y)  := edwards_mult hash dbl3_xout dbl3_yout in
  let '(right_x, right_y) := edwards_add R8x R8y right2_x right2_y in
  let '(left_x, left_y) := edwards_mult S q1 q2 in
  left_x = right_x /\ left_y = right_y.

Definition cons (enabled: F) (Ax: F) (Ay: F) (S: F) (R8x: F) (R8y: F) (M: F) :=
  exists (hash: @Poseidon.t 5) (eqCheckX eqCheckY: ForceEqualIfEnabled.t),
    (hash.(Poseidon.inputs)[0] = R8x) /\
    (hash.(Poseidon.inputs)[1] = R8y) /\
    (hash.(Poseidon.inputs)[2] = Ax) /\
    (hash.(Poseidon.inputs)[3] = Ay) /\
    (hash.(Poseidon.inputs)[4] = M) /\
    let '(dbl1_xout, dbl1_yout)  := BabyDbl Ax Ay in
    let '(dbl2_xout, dbl2_yout)  := BabyDbl dbl1_xout dbl1_yout in
    let '(dbl3_xout, dbl3_yout)  := BabyDbl dbl2_xout dbl2_yout in
    let '(right2_x, right2_y)  := edwards_mult hash.(Poseidon.out) dbl3_xout
    dbl3_yout in
    let '(right_x, right_y) := edwards_add R8x R8y right2_x right2_y in
    let '(left_x, left_y) := edwards_mult S q1 q2 in
    eqCheckX.(ForceEqualIfEnabled.enabled) = enabled /\
    eqCheckX.(ForceEqualIfEnabled._in)[0] = left_x /\
    eqCheckX.(ForceEqualIfEnabled._in)[1] = right_x /\
    eqCheckY.(ForceEqualIfEnabled.enabled) = enabled /\
    eqCheckY.(ForceEqualIfEnabled._in)[0] = left_y /\
    eqCheckY.(ForceEqualIfEnabled._in)[1] = right_y.
```

Specification and Proof The following shows the formal specification and proof for the `EdDSAPoseidonVerifier` template:

```
Lemma EdDSAPoseidonVerifier_spec :
  forall (p: @EdDSAPoseidonVerifier.t),
    p.(enabled) <> 0 ->
```



```
    eddsa_poseidon p.(Ax) p.(Ay) p.(S) p.(R8x) p.(R8y) p.(M).
Proof.
  intros. unwrap_C. destruct p eqn:hh;simpl in *.
  unfold cons in *.
  destruct _cons0. simpl in *. destruct p;simpl in *.
  destruct e.
  destruct e. pose proof a.
  intuition.
  destruct BabyDbl eqn: db1 in H6.
  destruct BabyDbl eqn: db2 in H6.
  destruct BabyDbl eqn: db3 in H6.
  destruct edwards_mult eqn: em in H6.
  destruct edwards_add eqn: ea in H6.
  destruct edwards_mult eqn: em2 in H6.
  intuition.
  subst.
  unfold eddsa_poseidon.
  rewrite db1, db2, db3.
  erewrite <- (Poseidon.PoseidonHypo.poseidon_5_spec x);eauto.
  rewrite em, ea, em2.
  pose proof (Cmp.ForceEqualIfEnabled.soundness x0).
  pose proof (Cmp.ForceEqualIfEnabled.soundness x1).
  simpl in *. subst. intuit.
  apply H1;auto. rewrite H8. auto.
Qed.

Definition unconstrained:= ForceEqualIfEnabled.unconstrained.

(* forall Ax Ay S R8x R8y M, EdDSAPoseidonVerifier 0 Ax Ay S R8x R8y M *)
Lemma EdDSAPoseidonVerifier_spec_disabled :
  forall Ax Ay S R8x R8y M (hash: @Poseidon.t 5) (eqCheckX eqCheckY:
    ForceEqualIfEnabled.t),
    (hash.(Poseidon.inputs)[0] = R8x) ->
    (hash.(Poseidon.inputs)[1] = R8y) ->
    (hash.(Poseidon.inputs)[2] = Ax) ->
    (hash.(Poseidon.inputs)[3] = Ay) ->
    (hash.(Poseidon.inputs)[4] = M) ->
    let
    '(dbl1_xout, dbl1_yout) := BabyDbl Ax Ay in
     let
     '(dbl2_xout, dbl2_yout) := BabyDbl dbl1_xout dbl1_yout
      in
      let
      '(dbl3_xout, dbl3_yout) := BabyDbl dbl2_xout dbl2_yout
       in
       let
       '(right2_x, right2_y) :=
        edwards_mult (Poseidon.out hash) dbl3_xout dbl3_yout
        in
        let
        '(right_x, right_y) :=
         edwards_add R8x R8y right2_x right2_y in
         let
```

```
56          '(left_x, left_y) := edwards_mult S q1 q2 in
57     eqCheckX.(ForceEqualIfEnabled.enabled) = 0 /\
58     eqCheckX.(ForceEqualIfEnabled._in)[0] = left_x /\
59     eqCheckX.(ForceEqualIfEnabled._in)[1] = right_x /\
60     eqCheckY.(ForceEqualIfEnabled.enabled) = 0 /\
61     eqCheckY.(ForceEqualIfEnabled._in)[0] = left_y /\
62     eqCheckY.(ForceEqualIfEnabled._in)[1] = right_y ->
63     unconstrained left_x right_x /\ unconstrained left_y right_y.
64 Proof.
65   intros.
66   destruct BabyDbl eqn: db1.
67   destruct (BabyDbl f _) eqn: db2.
68   destruct (BabyDbl f1 _) eqn: db3.
69   destruct edwards_mult eqn: em.
70   destruct edwards_add eqn: ea.
71   destruct (edwards_mult S0 _ _) eqn: em2. subst.
72   intuit.
73   pose proof (Cmp.ForceEqualIfEnabled.circuit_disabled eqCheckX);intuit.
74   rewrite <-H1,<-H. unfold unconstrained. auto.
75   pose proof (Cmp.ForceEqualIfEnabled.circuit_disabled eqCheckY);intuit.
76   rewrite <-H3,<-H5. unfold unconstrained. auto.
77 Qed.
```

5.1.6 V-SH2-SPEC-006: Poseidon is Deterministic

Commit	`0x2b79ab3`	**Status**	Verified
Files	poseidon.circom		
Functions	Poseidon		

Description The output of the poseidon(1) hash and poseidon(2) hash is deterministic. In other words, given the same inputs, the output of the Poseidon hash must be constant.

Informal Specification

$$\forall\ \text{inputs}, \text{out}_1, \text{out}_2.(\text{poseidon}(\text{inputs}) = \text{out}_1 \land \text{poseidon}(\text{inputs}) = \text{out}_2) \rightarrow \text{out}_1 = \text{out}_2$$

Proof This property was proved using Picus, an in house tool used to verify that ZK circuits are properly constrained. Picus proved Poseidon(1) and Poseidon(2) were properly constrained and since all properly constrained circuits are deterministic, we can safely conclude that Poseidon(1) and Poseidon(2) are deterministic.