ZKSECURITY

Audit of Self: Aadhaar Circuits

Date: September 9th, 2025

Introduction

On September 1st 2025, zkSecurity started a security audit of Self's Aadhaar-related Circom circuits. The audit lasted one week with two consultants. We reviewed the Self repository at commit 3905a30a.

This is zkSecurity's third audit collaboration with Self. The previous audits are available here: first audit and second audit.

Scope

We reviewed the following Circom circuits and all their new or updated dependencies:

- `REGISTER_AADHAAR`: verifies the authenticity of an Aadhaar QR code by checking its RSA signature; extracts structured user data; outputs a nullifier (deterministic identifier for uniqueness/Sybil-resistance) and a commitment (for Merkle registration without exposing personal data).
- `VC_AND_DISCLOSE_Aadhaar`: proves inclusion of the user's commitment in a known Merkle tree; optionally discloses selected Aadhaar fields (e.g., DOB, address) via a bitmap selector; checks whether name and DOB/YOB appear on an OFAC-like watchlist using Sparse Merkle Tree proofs.

As part of the dependency set, we also reviewed `pack.circom`, adapted from the Anon Aadhaar repository by PSE.

Finally, and independent of the Aadhaar focus of the audit, we reviewed a new generic, non-native field exponentiation circuit used for RSA verification:

- `FpPowGenericMod` at commit 145cc894

Overview

Aadhaar QR code format

At its core, the Aadhaar QR code contains a serialized data structure containing various identification fields, which is digitally signed by UIDAI (Unique Identification Authority of India) using RSA with SHA-256. The Aadhaar circuits target the "V2" version of this structure. Unfortunately, official documentation for the Aadhaar QR code format is limited, and a specification of the structure of the serialized data is not publicly available. The parsing and extraction of fields is therefore based on reverse-engineering and empirical testing.

The fields are separated by the byte `0xff` (255 in decimal). At the beginning, there are two constant bytes `"V2"` and then a separator byte `0xff`. After that, there are the following fields, each separated by a `0xff` byte:

1. Email and phone presence flags, can be 0, 1, 2 or 3

2. Reference ID, which is the last 4 digits of Aadhaar number and timestamp of the certificate generation
3. Name
4. Date of Birth
5. Gender
6. Address - Care of
7. Address - District
8. Address - Landmark
9. Address - House
10. Address - Location
11. Address - Pin code
12. Address - Post office
13. Address - State
14. Address - Street
15. Address - Sub district
16. Address - VTC
17. Last 4 digits of the mobile number

After the 18th `0xff` delimiter, it is stored the photo, up until the end of the data.

Nullifier design

The nullifier is a crucial component of the Self system, as it is used to prevent double-registration of the same document. Due to the fact that the full Aadhaar number itself is not included in the signed QR code, the nullifier must be derived from other fields. In the current design, the nullifier is computed as the Poseidon hash of the following fields:

- Name
- Date of Birth
- Last 4 digits of Aadhar number

All other fields are not used in the nullifier computation, as the user can somewhat freely change them (e.g., address, phone number) and generate a new signed QR code.

We make two general remarks about this design:

- As discussed in details in one finding below, it is not entirely true that the name field is immutable, as users can request changes to their name in the Aadhaar system with relatively low effort.
- The nullifier is not hiding, since it is derived from low-entropy fields. This means that, for example, an attacker who knows the full name of a target registered user can recover their date of birth using very little effort (e.g., by brute-forcing all possible dates of birth and all possible last 4 digits of Aadhaar numbers).

We note that this is a design decision that Self team is aware of, and is inherently a trade-off between usability and security. We are not aware of any way to significantly improve the design without having access to more stable and high-entropy fields, which are not available in the signed data.

Findings

Below are listed the findings found during the engagement. High severity findings can be seen as so-called "priority 0" issues that need fixing (potentially urgently). Medium severity findings are most often serious findings that have less impact (or are harder to exploit) than high-severity findings. Low severity findings are most often exploitable in contrived scenarios, if at all, but still warrant reflection. Findings marked as informational are general comments that did not fit any of the other criteria.

ID	COMPONENT	NAME	RISK
#00	country_not_in_list.circom	Forbidden Country Check Bypass via Packed Byte Overflow	High
#01	circuits	Invalid Assumptions About Aadhaar Name Field Formatting	High
#02	vc_and_disclose_aadhaar.circom	Missing Byte Range Checks Allows Packed Data Pollution	Medium
#03	extractQrData.circom	Photo Hash Depends on Non-Photo QR Data Length	Medium
#04	extractQrData.circom	Delimiter Ordering Not Enforced in ValidateDelimiterIndices	Low
#05	register_aadhaar.circom	Attestation ID Not Hard-Coded in Register Proof	Informational
#06	vc_and_disclose_aadhaar.circom	Dummy Constraint Is Linear and Optimized Away	Informational
#07	extractQrData.circom	Timestamp Rounding Comment Mismatch	Informational
#08	extractQrData.circom	Unused signal output `age` in `EXTRACT_QR_DATA`	Informational

#00 - Forbidden Country Check Bypass via Packed Byte Overflow

Severity: High **Location:** country_not_in_list.circom

Description. The `CountryNotInList` template checks that a given country code is not contained in an input list of forbidden countries. For Aadhaar, this is used with a hard-coded country code of IND (India) as input. The forbidden countries are packed into fewer field elements using `PackBytes`, and returned as public output.

```
template CountryNotInList(MAX_FORBIDDEN_COUNTRIES_LIST_LENGTH) {
    signal input country[3];
    signal input forbidden_countries_list[MAX_FORBIDDEN_COUNTRIES_LIST_LENGTH * 3];

    signal equality_result[MAX_FORBIDDEN_COUNTRIES_LIST_LENGTH][4];
    signal is_equal[MAX_FORBIDDEN_COUNTRIES_LIST_LENGTH][3];
    for (var i = 0; i < MAX_FORBIDDEN_COUNTRIES_LIST_LENGTH; i++) {
        equality_result[i][0] <== 1;
        for (var j = 1; j < 3 + 1; j++) {
            is_equal[i][j - 1] <== IsEqual()([country[j - 1],
forbidden_countries_list[i * 3 + j - 1]]);
            equality_result[i][j] <== is_equal[i][j - 1] * equality_result[i][j - 1];
        }
        0 === equality_result[i][3];
    }

    var chunkLength = computeIntChunkLength(MAX_FORBIDDEN_COUNTRIES_LIST_LENGTH * 3);
    signal output forbidden_countries_list_packed[chunkLength]  <==
PackBytes(MAX_FORBIDDEN_COUNTRIES_LIST_LENGTH * 3)(forbidden_countries_list);
}
```

The issue here is that elements of `forbidden_countries_list` are not range-checked to be bytes, which means `PackBytes` allows for aliasing. This weakness allows an attacker to easily bypass the intended forbidden country check, which has only the packed output bytes available.

For example, the exploit could be done as follows: Supply a list containing

- `[73, 78, 68 + 256]` (i.e., the three IND bytes, but adding 256 to the last byte);
- the bytes of some other country, modified by `-1` in the lowest byte.

The modification `+256` implies that the not-in-list check will succeed, because in unpacked form, none of the country codes equal IND. However, after packing, the country list encodes `[IND, <other country>]`. This allows the attacker to use an Aadhaar proof even if India is among the forbidden countries.

The same issue is present in forbidden country checks for passports and EU IDs, in the `ProveCountryIsNotInList` and `ProveCountryIsNotInList_ID` templates.

Impact. Forbidden country checks can be bypassed for all supported ID document types.

Recommendation. Add explicit byte range constraints, for example using `AssertBytes`, on every country code element before passing them to `PackBytes` in `CountryNotInList`, `ProveCountryIsNotInList`, and `ProveCountryIsNotInList_ID`.

Client Response. The issue was fixed in commits 60501d1 and 4914074.

#01 - Invalid Assumptions About Aadhaar Name Field Formatting

Severity: High **Location:** circuits

Description. Both the nullifier computation and the OFAC name check use the Aadhaar name fields as part of their inputs. The name field is not enforced by the issuing authority to be in a specific case (e.g., all uppercase), or in a specific format (e.g., first name first or last name first).

Additionally, users can request changes to the fields in their certificate, and there are some common cases where the name field can be changed with low effort (from the sources one and two). For example, a user can, upon request to UIDAI:

- Swap the order of their name components
- Change their name due to a marriage/divorce
- Expand/contract their initials, e.g., from the second source, "A person named as B I Hirani, would like to expand his first name or last name or both"
- Correct previous typos made by an operator during the creation of the document

Impact. Two critical components of the Aadhaar circuits rely on the name field:

- The computation of the nullifier, which is used to prevent double-registration and Sybil attacks.
- The OFAC name checks, which is used to ensure that the disclosing user is not on a sanctions list.

In both cases, a slight change in formatting or case of the name can lead to a completely different nullifier or a missed OFAC match.

Recommendation. We suggest the following mitigation strategy:

- Converting the name to uppercase when computing the nullifier.
- Converting the name to uppercase when performing the OFAC check.

This would at least prevent simple case changes from evading the checks, but would not prevent more complex changes such as reordering of name components or expansion/contraction of initials. Unfortunately, we are not aware of a concrete fix that would eliminate this issue, given the breadth of editing possibilities. We additionally remark that **this issue does not derive from any particular vulnerability in the circuits themselves**, but rather a limitation of the underlying Aadhaar system, making it inherently a weaker ID system than, e.g., passports, and could be inadequate when high confidence is required.

Client Response. The client acknowledged the issue, and implemented a sanity name uppercase conversion in commits a7f69f9 and c751abb, which mitigates simple case changes.

#02 - Missing Byte Range Checks Allows Packed Data Pollution

Severity: Medium **Location:** vc_and_disclose_aadhaar.circom

Description. In the `VC_AND_DISCLOSE_Aadhaar` circuit, `PackBytes` (from `@zk-email/circuits`) is used to pack the revealed data bytes. The template does not constrain each provided input value to be a byte, i.e. in the range $[0, 2^8)$. This allows crafting inputs exceeding 255. `isMinimumAgeValid` equals `minimumAge` when `age >= minimumAge`, which is done using `GreaterEqThan(7)([age, minimumAge])`. Note that `GreaterEqThan(7)([a, b])` passes even if b is not a 7-bit integer. For instance, `(a, b) = (1024, 1023)` or even `(a, b) = (1, 21888242871839275222246405745257275088548364400416034343698204186575808495616)` would pass the check.

Using a "negative" (large and close to the modulus) `minimumAge` enables pollution of the final packed output segment (bytes 93–118) that is expected to encode: part of the state, the last 4 digits of the phone number, OFAC result bits, and `minimumAge`.

The remaining bytes are passed to `PackBytesAndPoseidon`, in which each input is guaranteed to be a byte, thus the remaining portion of the packed bytes is unaffected.

Impact. The issue enables to disclose incorrect values for state and phone number. The possible pollution pattern is fixed. Flipping the OFAC check result from `false` to `true` is not possible, since the contract only accepts `1` as a truthy result.

Recommendation. Add an explicit range check using `Num2Bits(8)`, ensuring `minimumAge` is constrained to a byte.

Client Response. The fix was addressed in commit 285f0a9, where there is an additional range check on the `minimumAge` being a 7-bit value.

#03 - Photo Hash Depends on Non-Photo QR Data Length

Severity: Medium **Location:** extractQrData.circom

Description. The `PhotoExtractor` logic selects the first 31 * 32 bytes starting at the photo start index within the padded QR data, rather than isolating the exact JPEG/photo bytes. The extracted chunk therefore mixes:

- Actual photo bytes
- SHA-2 padding sequence: 0x80, zero bytes, and the 8-byte length field encoding total QR data length in bits
- Additional zero padding to pad the data to its in-circuit size

Because the 8-byte length field, and the amount of zero padding before it, changes whenever any *other* QR data length changes, the derived `photoHash` changes even if the underlying photo bytes are identical. The code comment suggests this variability was not intended:

```
// Pack byte[] to int[] where int is field element which take up to 31 bytes
// When packing like this the trailing 0s in each chunk would be removed as they are
LSB
// This is ok for being used in nullifiers as the behaviour would be consistent
```

Example extracted photo data (truncated) shows inclusion of padding and length field (bytes `128 0 .. 0 35 144` where `35 144` encodes the bit length):

```
[255,255,79,255,81,0,47,0,0,0,0,0,60,...,128,0,0,0,0,0,0,0,0,0,0,35,144,0,0,0,...,0]
```

Impact. `photoHash` is not a stable commitment to the photo alone; it can be altered by changing unrelated QR fields that precede the photo. This may undermine deduplication or linkage properties expected from a photo-derived identifier in applications built on top of Self.

Note that this does not impact the nullifier used by Self for registration, which does not contain the photo hash.

Recommendation. We helped Self developers to work out a solution, which relies on the fact that JPEG2000 images end with a two-byte marker `0xFF D9` which cannot appear anywhere else in the image bytes. The index of this marker can be passed in as an additional witness. To be sound, the index has to be validated in-circuit to

- be larger than the `photoPosition` index, and
- point to the two-byte sequence `0xFF D9`.

Extracting photo data only up to the end marker index addresses the issue.

Client Response. The fix described above was applied in commit eff1689 and validation was added in follow-up commits.

#04 - Delimiter Ordering Not Enforced in ValidateDelimiterIndices

Severity: Low **Location:** extractQrData.circom

Description. In the `ValidateDelimiterIndices` template, the outputs of the `LessThan` components are not enforced, so strict ordering of delimiter indices is not actually constrained:

```
delimiter_idx_less_than_nxt_idx[i] = LessThan(12);
delimiter_idx_less_than_nxt_idx[i].in[0] <== delimiterIndices[i];
delimiter_idx_less_than_nxt_idx[i].in[1] <== delimiterIndices[i + 1];
// [ZKSECURITY] we don't check that `delimiter_idx_less_than_nxt_idx[i].out === 1`
```

And again on the final check:

```
component is_last_delimiter_idx_valid = LessThan(12);
is_last_delimiter_idx_valid.in[0] <== delimiterIndices[17];
is_last_delimiter_idx_valid.in[1] <== maxDataLength;
// [ZKSECURITY] we don't check that `is_last_delimiter_idx_valid === 1`
```

Impact. We believe it is not possible to exploit the missing validation, since delimiters are either unused, or indirectly constrained to valid positions, or at least constrained to a correct value modulo `maxDataLength` which does not have an impact on circuit correctness. However, this analysis is quite subtle, because without the validation, input assumptions of most calls to `SelectSubArray` are violated.

Recommendation. For robustness and simpler security analysis, we recommend to keep the checks and fix them by constraining `LessThan` outputs to be `1`.

Client Response. The issue was fixed in commit 2dcb67d

#05 - Attestation ID Not Hard-Coded in Register Proof

Severity: Informational **Location:** register_aadhaar.circom

Description. A register proof can currently be generated with an `attestation_id` value different from the expected Aadhaar value (3). The system will accept and register the resulting commitment.

Impact. While contracts use the attestation ID to select the appropriate proof type, the structure of Aadhaar proofs is so different from other proofs that one will not pass as the other. Therefore, we deem this to have no security impact in practice: a registered commitment with ID not equal to 3 would be unusable for disclosure.

Nevertheless, this could become an issue if, for example, another version of Aadhaar would be added.

Recommendation. Hard-code `attestation_id = 3` inside the register circuit, so proofs with divergent values cannot be produced or accepted.

Client Response. The issue was fixed in commit c751abb.

#06 - Dummy Constraint Is Linear and Optimized Away

Severity: Informational **Location:** vc_and_disclose_aadhaar.circom

Description. The disclose circuit puts a dummy constraint on the `user_identifier`:

```
signal dummy <== user_identifier + user_identifier;
```

This constraint was added for replay protection, which is not necessary since Circom always includes R1CS constraints for public inputs like `user_identifier`.

If such a constraint were required, it would need to be nonlinear, because linear constraints are optimized away when using the `--02` flag. In that case, `+` should be replaced with `*`.

Client Response. The dummy constraint was removed in commit 3ba0ea4.

#07 - Timestamp Rounding Comment Mismatch

Severity: Informational **Location:** extractQrData.circom

Description. The `TimestampExtractor` comment claims minutes and seconds are ignored (rounded to the nearest hour), but the circuit uses the provided `minute` value and only sets `second` to zero.

```
/// @notice Extracts the timestamp when the QR was signed rounded to nearest hour
/// @dev We ignore minutes and seconds to avoid identifying the user based on the
precise timestamp
/// ...
template TimestampExtractor(maxDataLength) {
    // ...
    component dateToUnixTime = DigitBytesToTimestamp(2032);
    dateToUnixTime.year   <== year;
    dateToUnixTime.month  <== month;
    dateToUnixTime.day    <== day;
    dateToUnixTime.hour   <== hour;
    dateToUnixTime.minute <== minute;
    dateToUnixTime.second <== 0;

    timestamp <== dateToUnixTime.out - 19800; // 19800 is the offset for IST
}
```

Recommendation. Align comment and behavior: either change the comment to state that only seconds are discarded, or modify the circuit to ignore minutes as well if hour-level rounding is intended.

Client Response. The comment was corrected in commit 7eeccd5. The Self team clarified that minutes are needed to check that the Aadhaar ID was signed within the last 20 minutes.

#08 - Unused signal output `age` in `EXTRACT_QR_DATA`

Severity: Informational **Location:** extractQrData.circom

Description. The `EXTRACT_QR_DATA` circuit exposes a `signal output age` that is never assigned and is not used elsewhere.

Client Response. The unused signal was removed in commit e8f8093.