Zero Knowledge Barcode Decoding with Application to Private Online Attribute Verification

Kelsey Merrill Anna Woo Wenting Zheng Sarah Scheffler

Carnegie Mellon University

Abstract

Online attribute checking (e.g. proving age, residency) is increasingly common, yet standard implementations reveal far more personal information than necessary (e.g. all ID contents). Privacy-preserving alternatives exist but require digital inputs: anonymous-credentials or zero-knowledge (ZK) proofs of signature possession over a bitstring. However, it is challenging to gain integrity guarantees on the bitstring itself.

C2PA offers a partial solution: C2PA-enabled cameras cryptographically attest to image origins with an embedded signing key, so a smartphone could provide a signed image of an ID barcode. However, since C2PA signs the image rather than the bitstring of the decoded barcode, the prover must additionally prove correct execution of the PDF417 barcode decoding algorithm on the signed image. Two barriers block this approach: images are large, yielding large proofs and long prover runtimes, and the PDF417 barcode decoding algorithm is highly data-dependent, making compilation into a ZK-friendly constraint systems non-trivial.

We present an end-to-end ZK proof system for PDF417 barcode decoding, built on an adaptation of zkSNARK system Dorian [1] (itself based on Spartan [2]) with modifications: (1) adjusting Dorian's polynomial commitment to validate C2PA signatures more efficiently while cheaply checking consistency with the main Dorian proof, and (2) incorporating additional technical gadgets for set disjointness, data-dependent processing in R1CS, and state machines for greater efficiency. Implementing the PDF417 decoding algorithm as R1CS constraints is also nontrivial, as the algorithm is highly data-dependent and requires modifications to ensure soundness.

Our system is the first to enable efficient barcode decoding in ZK. The best previous option was a zkVM, requiring prohibitively high computation and memory. We demonstrate that our system is significantly faster and uses far less memory. Furthermore, we suggest changes to the C2PA framework that would make future private verifiable image processing tasks more efficient. Though not yet ready for practical deployment, our system presents an alternative approach to private online attribute verification and demonstrates techniques of independent interest for data-dependent ZK computation.

1 Introduction

Users are sometimes asked to verify personal attributes online using a government-issued identity document (ID), for example, age, residency, student status, or credential validity. Online age verification is of particular interest, as regulators in the US, UK, EU, and elsewhere are requiring websites that serve age-restricted content to conduct age verification on their clients [3–5]. Several forms of age assurance exist; many US states indicate a preference or mandate for age verification based on government-issued ID cards (see e.g., [6, 7]). However, these methods pose privacy and security concerns due to the massive transmission of users' identity information [8, 9].

To enable document-based age verification without transmission of sensitive data, a line of work on cryptographic zero-knowledge age verification has developed [1, 10–13]. These systems work by having a user's device act as a Prover, with the user's ID information in bitstring form serving as the witness. The Prover proves in zero knowledge some function of the input bitstring, typically indicating that (1) the user meets the age requirement, and (2) the ID credentials contain a valid signature from the ID issuer. The Verifier is able to check these properties but does not learn the user's other ID information.

These systems prove that the user possesses *a bitstring corresponding to a valid ID*, but they do not address *the integrity of the bitstring itself*. In the absence of comprehensive digital ID infrastructure, however, these works have no verified way to obtain the bitstring from a physical ID card; they take the bitstring as given. This work explores one method for closing that gap.

An opportunity arises from the proliferation of trusted cameras that digitally sign images for later verification under the Coalition for Content Provenance and Authenticity (C2PA) standard [14, 15]. For example, Google's Pixel 10 smartphone released in August 2025 uses C2PA image signing [16]. Using a trusted camera, a user can obtain a cryptographically-verified photograph of their physical ID card, which can then be decoded under zero knowledge. A trusted camera takes a photo and signs it, attesting to the image's authenticity. Then, the user's device processes the image under zero knowledge to retrieve the information stored in the ID's barcode as a bitstring. From there, existing techniques can be used to go from this bitstring to a full proof of the desired attribute [1, 10–13].

Using this zero-knowledge proof, the verifier can confirm both that the barcode decoding was performed correctly and that the barcode in the picture has the claimed attribute. This method protects all of the user's information except the specific attribute they are trying to share. Additionally, doing verification like this allows us to prove that a user is able to take a recent photo of an ID, providing evidence that they are a real person and not a bot—an advantage lacking in proofs relying solely on digital credentials.

Like any photo-based approach to confirm digital credentials, our system cannot prevent a user from photographing a copy of a barcode rather than the original—however, it does confirm that the image came from a trusted camera. The trusted camera provides an improvement over the assurance of uploading a traditional photo (the current status quo), and remains harder to subvert than systems that rely solely on stored digital credentials. Our approach is also flexible with respect to hardware: it requires no specialized chips in the physical ID itself, and unlike architectures that push heavy computation into a trusted-execution environment (TEE), the only possible use of trusted hardware is as the C2PA root of trust. This hardware flexibility is important because many devices have small TEEs with limited computational power and memory.

Turning to technical challenges of extracting encoded barcode data from an image of the barcode under zero knowledge, existing solutions are not sufficient. As discussed in prior work [17, 18], off-the-shelf zkSNARKs are insufficient for image processing because images are very large compared with the typical input to zero-knowledge proof systems. However, even tailored zero-knowledge image-processing solutions are insufficient for this task because the PDF417 barcode decoding algorithm is extremely *data-dependent*—the structure of the computation depends on the secret data being computed on. This sort of computation is traditionally difficult to translate to zero-knowledge; in the commonly-used arithmetic circuit model (or its generalization, R1CS), the structure of the computation must be public, but with such algorithms, the structure depends on the data we wish to keep secret. In fact, even prior work on image processing under zero knowledge does not address such algorithms, instead focusing on operations that are data-independent and parallelizable, like translations and crops with public boundaries [17–19].

1.1 Our contributions

In this paper, we present:

1. **An end-to-end method for reading a PDF417 barcode from an image under zero knowledge** using the Spartan-based Dorian zkSNARK system with modifications (overview in Section 3, details in Section 5). Rather than directly translating the PDF417 decoding algorithm to R1CS, we designed an R1CS version of the algorithm to efficiently handle its data-dependent elements To do this, we introduced prover "advice" (additional private witness elements) with corresponding soundness checks to catch malicious inconsistencies. We also offer an implementation and benchmarks (Section 6): our system is more than 18x faster and uses 6.9x less RAM than the next-best alternative for zero-knowledge barcode decoding.

2. **Technical gadgets of general interest** (Section 4).

   - Applications of lookup tables and prover advice to circumvent traditional issues with data-dependent algorithms under zero knowledge, such as dynamic indexing, state machines, and conditional computation (Sections 4.1, 4.2).
   - A modification to the Spartan proof system to allow the prover to avoid verifying a signature inside of the SNARK, instead pushing this check to the verifier (Section 5.9).

3. **Suggestions for modifying C2PA** to better enable zero-knowledge-friendly computation (Section 8). We utilize a modified version of C2PA in our system, and we suggest changes to the spec that would allow for more private post-processing.

Though our system for decoding PDF417 barcodes from images is not yet ready for practical deployment, this is the first work that we know of that attempts this kind of data-dependent computation outside of a zkVM. With our specially-designed constraint system, we achieve 18.6x speedup over a zkVM with 6.9x less memory.

The rest of the paper proceeds as follows. Section 2 contains necessary technical preliminaries. Section 3 gives an overview of our system and details our goals and assumptions. Section 4 describes techniques we developed that are of general interest for data-dependent processing under zero knowledge. Section 5 details our system's prover protocol. Section 6 reports our experimental analysis. Section 7 discusses how we meet our security goals. Section 8 proposes changes to C2PA. Section 9 compares our system to prior work. Finally, Section 10 discusses directions for future work and concludes the paper.

2 Preliminaries

2.1 Cryptography Preliminaries

A **digital signature** is a set of three algorithms (`KeyGen`, `Sign`, `Verify`) that is EU-CMA secure in the standard way [20, Section 12.2].

A **commitment scheme** is a set of two algorithms (`Commit`, `Open`) that has computational hiding and binding properties in the standard way [20, Section 5.6.5]. A commitment scheme is called *extractable* if there is an efficient algorithm (an "extractor") that can retrieve the commitment pre-image given the description of the committer [21].

A **polynomial commitment scheme** (PCS) is a set of three algorithms (`Commit`, `Open`, `Eval`) that also has computational hiding and binding properties [22]. A polynomial commitment scheme may also be extractable.

2.2 C2PA

C2PA, or the Coalition for Content Provenance and Authenticity, is a committee of companies that have created a specification for establishing image origin and edit history [14]. It is meant to be implemented in cameras and image editing software. It works by having the camera sign the image it captures along with associated metadata, thus attesting to the image's authentic creation by that camera.

The camera first computes a hash over image and metadata, and then it signs this hash. To make later make edits, the image editing software records the hash of the original image and the edits that it made, and it signs this record along with the output pixels. To verify the authenticity of an image, the record is checked for consistency, and the signature over the hash of the image and the record is verified against the camera's public key. The C2PA spec requires that SHA2 be used as the hash function and either ECDSA, RSA, or Ed25519 be used for the digital signature.

For the purposes of this paper, we generalize C2PA. Instead of a hash-then-sign protocol, we define a commit-then-sign protocol that also has an *open* algorithm for the commitment scheme. We also ignore C2PA metadata as it is not used in barcode decoding, but it could be used in an additional proof to verify the consistency of the barcode image with timestamps or other metadata information. We define C2PA as four algorithms:

- $\texttt{C2PA.Digest}(I, r) \to d$ - Compute a commitment over the image I using randomness r.
- $\texttt{C2PA.DigestVer}(I, d, r) \to \{0, 1\}$ - Check digest d against randomness r and image I.
- $\texttt{C2PA.Sign}(sk_{cam}, I, r) \to (d, \sigma)$ - Compute the digest for I using r and then sign it.
- $\texttt{C2PA.Verify}(pk_{cam}, \sigma, I, r)$ - Compute the digest for I using r and then check that σ is a valid signature over the digest according to pk_{cam}. For our purposes, the verifier will actually take in the digest d as an input and verify the signature directly (see Section 3.1).

We assume the existence of a C2PA device public key registry for verification. To avoid revealing the individual device that did the signing, we assume a ring signature over many keys in the registry. We believe this is consistent with many intended use cases of C2PA to verify validity of an image as coming from, e.g. "a valid Google phone," without revealing *which* phone. See Section 8 for further discussion.

2.3 Zero Knowledge, Proof-of-Knowledge, and zkSNARKs

A zero-knowledge proof (ZKP) is a cryptographic primitive that allows a *prover* to convince a *verifier* that there exists some (x, w) pair satisfying relation $\mathcal{R}$ for a known x, without revealing any information about w. The proof must be complete (all $(x, w) \in \mathcal{R}$ should verify), sound (no $(x, w) \notin \mathcal{R}$ should verify), and zero-knowledge (the verifier does not learn anything about w)—except with perhaps negligible probability in all cases. Crucially, the prover does not need to actually know w, only that it exists.

A zero-knowledge proof-of-knowledge (ZK-PoK) is a strengthening of a regular ZKP that requires the prover to actually *know* the secret witness w. ZK-PoK's have the additional property of *knowledge soundness*: a prover should not be able to convince a verifier unless it "knows" w, except with perhaps negligible probability. Formal definitions can be found in Appendix A.

2.3.1 zkSNARKs

Zero-knowledge succinct non-interactive arguments of knowledge (zkSNARKs) are proof systems that produce short, efficiently-verifiable statements about some secret witness. For some public input x, secret witness w, and binary relation $\mathcal{R}$, a zkSNARK can show $(x, w) \in \mathcal{R}$. A zkSNARK consists of three algorithms: (G, P, V).

- $G(1^\lambda) \to pp$ - the generator computes the public parameters based on the security parameters λ.
- $P(pp, x, w) \to \pi$ - the prover computes a proof π based on the public parameters, the public input, and the secret input.
- $V(pp, x, \pi) \to \{0, 1\}$ - the verifier checks the proof against the public input and the public parameters, and it outputs either accept (1) or reject (0).

A zkSNARK must have all the properties of a zero-knowledge proof-of-knowledge as well as *succinctness* - the proof must be short compared to the witness.

2.3.2 R1CS

Rank one constraint system satisfiability (R1CS-SAT) is an NP-complete language used by many zkSNARKs. For $n, m \in \mathbb{N}$, $A, B, C \in \mathbb{F}^{n \times m}$, a tuple $(A, B, C) \in \mathcal{L}$ iff $\exists z \in \mathbb{F}^m$ such that $z \neq \mathbf{0}$ and $(A \cdot z) \circ (B \cdot z) = (C \cdot z)$. A binary relation for a zkSNARK is constructed from R1CS-SAT as follows: $(x, w) \in \mathcal{R}_{A,B,C}$ iff, when $z = 1||x||w$, $(A \cdot z) \circ (B \cdot z) = (C \cdot z)$. Interactive R1CS (I-R1CS) is a version of R1CS that allows for randomized computations with verifier challenges [23].

2.3.3 Spartan

Spartan [2] is a zkSNARK that works over R1CS. It is a combination of the Spartan IOP and a generic extractable polynomial commitment scheme. It achieves linear proving time, and as long as the PCS does not require trusted setup, the whole SNARK also does not require trusted setup. Likewise, if the PCS is post-quantum, the whole SNARK is plausibly post-quantum secure.

The Spartan paper presents a useful optimization to reduce the amount of data the prover must commit to. Spartan uses multilinear polynomial extensions (MLE) of matrices A,B,C and vector Z to represent R1CS as a polynomial. First, matrices/vectors are represented as functions where the input is the row/column selector and the output is the value in that position. Then, the functions are converted to polynomials via MLE.

Committing to the entire polynomial representation $\tilde{Z}$ of $z = x||1||w$ is wasteful since the verifier knows all of that information already except for w. Instead, the prover just commits to $\tilde{w}$, and the verifier can compute $\tilde{Z}$ as follows:

$$\tilde{Z}(r) = (1 - r[0]) \cdot \tilde{w}(r[1...]) + r[0] \cdot \tilde{(x, 1)}(r[1...])$$

Assume $|w| = |x| + 1$ - if this is not the case, either w or x can be padded such that it is. Then, recall that $\tilde{Z}$ is a polynomial extension of a function over the boolean hypercube. The first entry in the vector r selects which half of z to look in. Assuming the vector entries are booleans, this formula looks at the first entry of r as a "selector bit" to decide which function to draw from. In the MLE version, the vector r is a vector of field elements, not booleans. However, by the uniqueness of the MLE, there must be only one MLE that agrees with a given function over the boolean hypercube. So, we can use the exact same selector terms, even with inputs that are not on the hypercube.

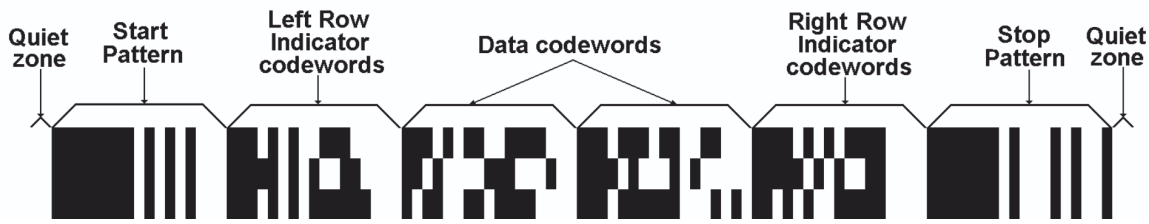


Figure 1: PDF417 structure modified from ISO 15438 [25], Section 5.2.1

2.3.4 Lookup arguments

Lookup arguments are proof systems for a particular relation. Given a public set $T \in \mathbb{F}^{|T|}$ and a private multi-set $A \in \mathbb{F}^{|A|}$, $(T, A) \in \mathcal{R}$ iff $\text{supp}(A) \subseteq T$. In this work, we build on the lookup arguments presented in [24]. It uses a randomized equality check to show

$$\sum_{i=0}^{|A|} \frac{1}{x - A[i]} = \sum_{j=0}^{|T|} \frac{e_j}{x - T[j]}$$

for prover-provided e_j's. Completeness and soundness are detailed in [24], and the technique requires $|A| + |T|$ constraints. [24] also extends this argument to work for tuple-valued lookups.

2.3.5 Dorian

Dorian [1] is a zkSNARK for I-R1CS. It adapts Spartan to the I-R1CS setting, allowing for lookup arguments and other tests requiring a verifier challenge.

2.4 PDF417 Barcodes

A PDF417 barcode [25] is a 2D barcode used on many identification documents around the world. It is part of the American Association of Motor Vehicle Administrator's standard for driver's licenses and ID cards [26]. Data is encoded in rows of black and white patterns. Barcodes have 3 to 90 rows and 1 to 30 data columns, and they use Reed-Solomon error correction. The error correction level `ec_level`, between 0 and 8, determines the number of error correction codewords (computed as $2^{\texttt{ec\_level}+1}$).

Each row has the structure: START symbol, left row indicator, data columns, right row indicator, STOP symbol (see Figure 1). Row indicators encode barcode metadata, such as the number of rows, the number of columns, and the error correction level. The very first data codeword in the barcode is the *symbol length descriptor* (SLD) which tells how many data codewords and pad codewords are in the barcode, including the SLD itself.

For our purposes, the decoding algorithm will also output auxiliary information to serve as advice to the prover.

There is no single canonical PDF417 decoding algorithm. The PDF417 spec provides a reference algorithm [25, Annex J], but it leaves many details unspecified. Also, common implementations also often stray from the algorithm presented in the specification for more tolerance to scanning errors or image artifacts [27,28]. Thus, it is possible that two different valid decoding implementations might yield different outputs for the same input. To account for this, we define two related algorithms:

- PDF417.`VerifyDecode`$(I, x) \to \{0, 1\}$: Returns 1 if (I, x) is a valid image-decoding pair, 0 otherwise.

- $\mathsf{PDF417.Decode}(I) \to (x, aux)$: Returns a valid decoding x and auxiliary prover information aux. We require $\mathsf{PDF417.VerifyDecode}(I, \mathsf{PDF417.Decode}(I)) = 1$ for all I. Any valid decoding algorithm may be used; we use a modification of rxing [29].

3 Setting and Security Goals

3.1 Overview

Our setting includes three actors:

- A **generator**, that generates the camera's key pair and the public parameters for proving. In practice, the party that generates the camera's key pair and the proof public parameters can be different, but we model this as a single algorithm: $G(1^\lambda) \to (pp, sk_{cam}, pk_{cam})$.
- The **prover**, who wishes to demonstrate possession of an authentic image of a PDF417 barcode encoding some known data. The prover is modeled as an algorithm P that accepts an image, its decoding, and some auxiliary information and outputs a proof: $P(I, x, aux) \to \pi$.
- The **verifier**, who wishes to verify that some known data came from an authentic image of a PDF417 barcode. The verifier is modeled as an algorithm V that accepts a digest, a signature, a digital signature public key, a decoding, and a proof, and it outputs accept or reject: $V(d, \sigma, pk_{cam}, x, \pi) \to \{0, 1\}$. We assume the verifier receives the legitimate pk_{cam} according to some C2PA public key registry (Section 2.2).

At a high level, the protocol proceeds as follows.

1. The generator generates sk_{cam}, pk_{cam}, and pp.
2. The prover receives an image I, digest d, and signature σ from a C2PA-enabled camera (see Section 2.2).
3. The prover decodes the barcode in I to data x.
4. The prover generates a zero-knowledge proof of knowledge (see Section 2.3) π of an image containing a barcode encoding such data.
5. The prover sends the digest d, signature σ, the encoded data x, and the proof π to the verifier.
6. The verifier checks that σ is a valid signature over digest d according to the camera's public key.
7. The verifier checks that proof π is valid.

We define the following security goals. Formal definitions can be found in Appendix B.

- **Completeness** - authentic images with valid decodings should always verify.
- **Soundness against inauthentic images** - images not captured by the C2PA-enabled camera should not be accepted by the verifier.
- **Knowledge soundness** - A prover that convinces the verifier must know an image consistent with the digest d containing a barcode that decodes to the data x.
- **Zero knowledge** - The verifier should not learn anything about the image except that it contains a PDF417 barcode decoding to x.

3.2 Assumptions and Limitations

We make the following assumptions. Removing these assumptions is left for future work.

3.2.1 We make public and assume constant the image coordinates of the barcode

This requires new set up for each image, which is not practical, but a common choice among other work (e.g. VerITAS and VIMz crop [17, 18]). It also has consequences for the zero knowledge property. For example, consider decoding barcodes from a driver's license as in the online age verification case. Different states place the barcode in different positions on the ID. Since users are likely to take pictures with the ID approximately centered in the image, these coordinates could give the verifier information about which state likely issued the ID.

3.2.2 The image has no rotation or skew

We assume that the rows of pixels are exactly parallel with the logical rows of the barcode, and resolution is constant across the barcode. While this is not realistic in many real-world settings, we leave dealing with skew to future work.

3.2.3 The decoding doesn't have "too many" errors

PDF417 barcodes use Reed-Solomon error correction to correct errors that may happen during scanning and decoding (see Section 2.4). Our proof system checks that this decoding has been done correctly, but our check is slightly too strict. Let s be the number of substitutions and t be the number of erasures. We require that $\texttt{num\_ec\_cw} - 2(s+t) - 3 \geq 0$, but actually only $\texttt{num\_ec\_cw} - 2s - t - 3 \geq 0$ is required, or in some cases, $\texttt{num\_ec\_cw} - 2s - t - 2 \geq 0$. As a result, our system will report invalid error correction for some barcodes that near the maximum number of errors correctable. We use this stricter check because it is more amenable to representation in R1CS.

Note that barcodes that have too many errors to decode correctly using the full RS error correction algorithm will still fail to decode (e.g. soundness holds). It is only that some barcodes right on the edge of being decodable will not validate. This only affects completeness, and if a particular photo has too many errors, a user could retake the photo.

3.2.4 The barcode is encoded entirely in text mode

PDF417 barcodes can encode data in three different *modes*, either "text," "binary," or "numeric." Encoding can switch between modes within one barcode using special codewords. We assume that the entire barcode is encoded using text mode, and the encoding never switches to binary or numeric mode. We focus on "text" mode because it is the likely setting for most PDF417 barcodes that would be subject to further proof (including in the identification document setting). We do not include the other modes because they add additional data-dependent processing complexity that would slow down the proof and provide no benefit to text-mode barcodes like we expect for this application.

Limitations

As implemented, our system focuses on the barcode decoding and outputs the barcode directly. To integrate with a full credential check, our system must be combined with the credential checking ZK proof, for example those described in Section 9. The actual ZK decoding process is also limited

by the assumptions given in Section 3.2. Further, we must be clear that no system such as this can never fully solve the questions of liveness or holder-binding in ID checks. We claim only that it is more sound than the alternative of relying solely on the digital credential check. This method may be used alongside other checks on biometrics, location, or other more invasive methods if such assurance is necessary.

4 Generalized Techniques and Gadgets

In this section, we present methods and techniques of general interest that are useful when expressing a data-dependent algorithm in an arithmetic circuit or R1CS. We will use these techniques throughout Section 5.

4.1 Prover advice with randomized consistency check for data-dependent algorithms

Data-dependent algorithms often determine which data they compute on in the future based on other data. However, the structure of the arithmetic circuit (or constraint system) must be constant and public. We circumvent this issue with prover advice and a randomized consistency check.

Consider an array A of inputs and array B outputs for some sub-computation that determines which data from A will be used in some other sub-computation. In order to preserve zero knowledge, we must have $|A| = |B|$, otherwise it would expose the number of elements from A selected. Naively, one might initialize B with some pad value and loop as in Algorithm 1.

Algorithm 1 Naive data-dependent selection

```
ind ← 0
for i ← 0 to |A| − 1 do
    if should_use(A[i]) then
        B[ind] ← A[i]
        ind ← ind + 1
    end if
end for
```

However, there are two issues that make this difficult to translate into an arithmetic circuit. First, there must be something written to B at each step in the loop to preserve zero knowledge; otherwise, the structure of the circuit reveals which elements of A are written to B. Secondly, the position at which the write occurs must be a constant. [23] presents a way to do dynamic array writes like this, but the per-write cost is still expensive.

We can fix these issues by defining some dummy value (perhaps the same as the pad value above, but not necessarily) and instead looping like in Algorithm 2.

With this loop, we write something to a constant location in B on each step, eliminating the issues above. However, now the real data we care about is interspersed with dummies. We can remove these dummies by having the prover provide the desired output with no dummies and show consistency between our computed B and this prover-provided array using a randomized polynomial identity test.

Call the prover-provided array C. Note that $|C| = |B| = |A|$, and C also makes use of pads as needed. Compute polynomials $b(x)$ and $c(x)$ and assert they are equal as shown in Algorithm 3.

This method requires $3 \cdot |A| + 5 \cdot |A| + 2|A| + 1 = 10 \cdot |A| + 1$ constraints. The first loop computing B requires 3 constraints per iteration for the if-else. The second loop computing b requires 5

Algorithm 2 Circuit-compatible data-dependent selection

```
for i ← 0 to |A| − 1 do
    if should_use(A[i]) then
        to_write ← A[i]
    else
        to_write ← DUMMY
    end if
    B[i] ← to_write
end for
```

Algorithm 3 Polynomial encoding for consistency check

```
pow ← 1,    b ← 0
for i ← 0 to |A| − 1 do
    if B[i] = DUMMY then
        elt ← 0
        pow_mult ← 1
    else
        elt ← B[i]
        pow_mult ← x
    end if
    b ← b + elt · pow
    pow ← pow · pow_mult
end for
pow ← 1,    c ← 0
for i ← 0 to |A| − 1 do
    c ← c + C[i] · pow
    pow ← pow · x
end for
assert b == c
```

constraints per iteration - 3 for the if-else, and 2 to update b and `pow`. The third loop computing c requires 2 constraints per iteration - just the updates to c and `pow`. One additional constraint is required to show equality. We measured the optimized constraint count with CirC [30], and we empirically saw approximately $10 \cdot |A|$ constraints needed for this method. Using the techniques in [23, Table 2] to do dynamic array accesses, we measure this requires approximately $40 \cdot |A|$ constraints.

4.2 Lookup tables for state machines

Lookup tables can be used to show the correct processing of state machines. Construct a table of all valid state machine transitions; i.e., elements of the table are tuples (s_i, s_{i+1}) for all such valid pairs. The prover provides a list of all states it moves through during the computation. It then shows that each transition is valid by showing a lookup into the table of (s_i, s_{i+1}) for all i (see Section 2.3.4).

As long as the start state is consistent with the computation's inputs, the end state is consistent with the computation's outputs, and all transitions are contained in the lookup table of valid transitions, the prover must have indeed moved through the state machine correctly.

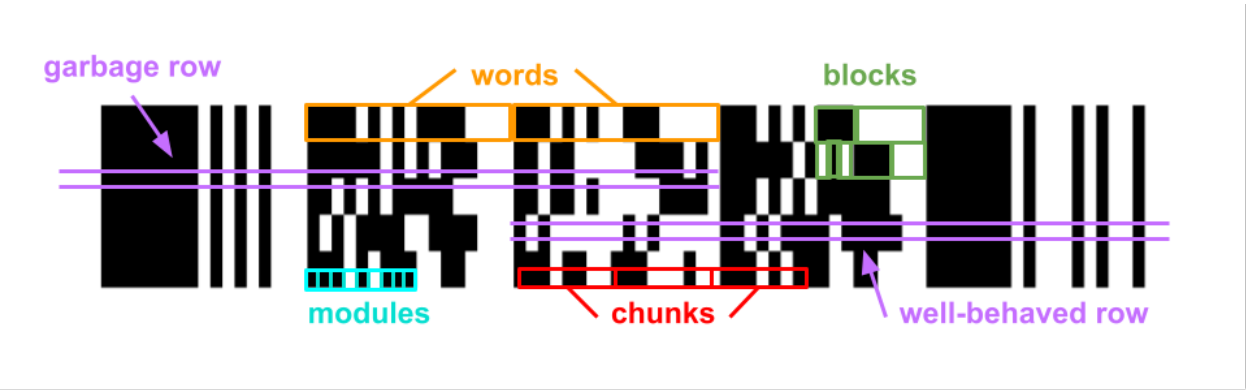


Figure 2: Decoding vocabulary

5 Prover Protocol for Barcode Decoding

The prover takes as input the image and the corresponding polynomial commitment produced by the camera. Then, the prover uses a slightly modified version of Spartan to prove knowledge of an image containing a barcode that decodes to the encoded data.

Constraint system. The constraint system contains eight sub-circuits for PDF417 decoding (Figure 3):

1. **Binarize** - convert the image from grayscale to black-and-white. (Section 5.1)
2. **Block measurement** - within each row, measure each continuous run of black or white in pixels. (Section 5.2)
3. **Normalization** - convert block lengths to *modules* rather than pixels; combines modules into *words* such that 8 consecutive blocks are contained within single field elements (Section 5.3)
4. **Codeword interpretation** - interpret PDF417 codewords from the words. (Section 5.4)
5. **Barcode metadata** - retrieve barcode metadata encoded in row indicators. (Section 5.5)
6. **Error correction** - run BCH Reed-Solomon error correction. (Section 5.6)
7. **Barcode size** - check that the barcode size and the amount of data found is consistent with the metadata encoded in the barcode. (Section 5.7)
8. **Character interpretation** - interpret the corrected codewords as ASCII characters. (Section 5.8)

Consistency between Proof and Digest. In addition to proving the correctness of the decoding, the prover must also show the consistency of the input witness with the committed C2PA digest (Section 5.9).

5.1 Binarize

Binarization converts the C2PA-signed grayscale image into black-and-white to make subsequent steps more efficient. We do this with range checks (see Appendix C.1).

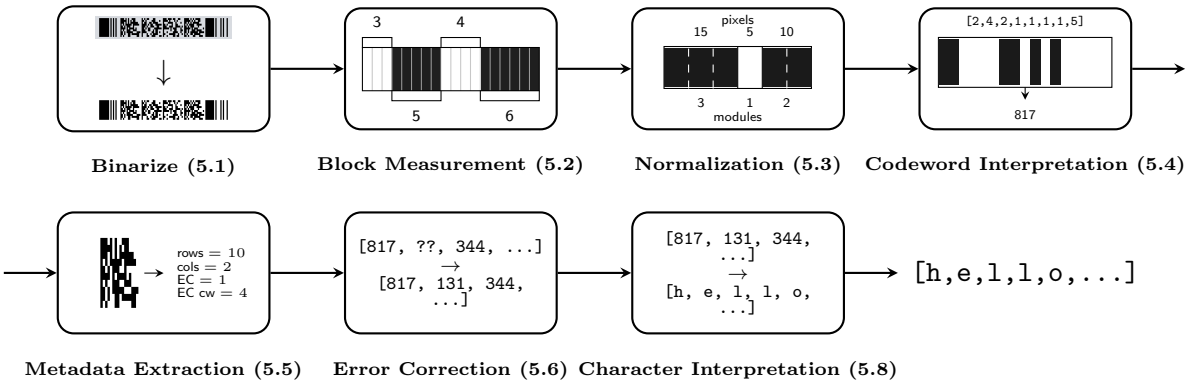


Figure 3: Decoding algorithm overview

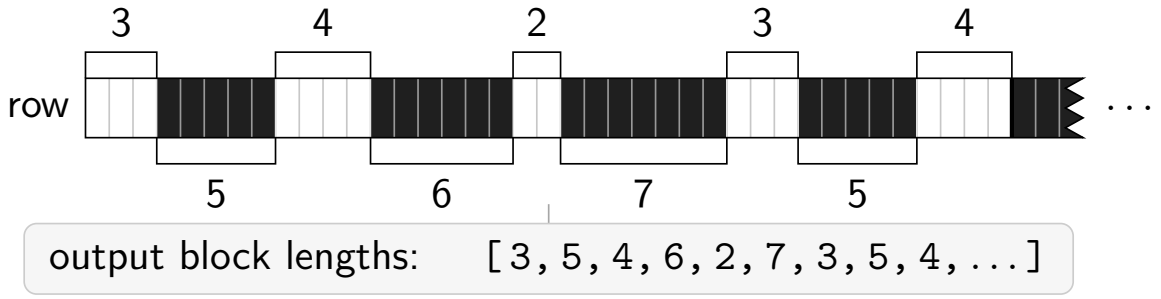


Figure 4: Block measurement in a PDF417 barcode row

5.2 Block measurement

The goal of block measurement is to count the number of pixels in each *block* (a run of pixels of the same color; see Figure 2). These counts will later be converted into PDF417 codewords. See Figure 4 for an overview.

Given a barcode represented as a binary 2D-array of total flattened length $p = \mathtt{R} * \mathtt{C}$, we count the lengths of consecutive white and black pixels in each row. Crucially, the number of blocks in each row is not public or constant; it depends on how many rows and columns the particular barcode has. Although the exact number of blocks can reveal information about the barcode we wish to keep secret, we do assume a publicly-known upper bound on block length (B) and on the maximum number of blocks in one row (M). These values can be computed from the dimensions of the barcode in the image (public in our setting—see Section 3.2.1) and the PDF417 spec.

Simply iterating over the pixels and accumulating them into blocks is unfriendly in R1CS; it requires $O(p)$-many expensive dynamic array indexing operations and/or conditionals. A second naive approach would look up the entire barcode row in a lookup table, but this requires $2^{O(p)}$ table entries, resulting in prohibitive lookup costs. Instead, we introduce a public parameter $\mathtt{L}$, decompose each row of the barcode into $\lceil \mathtt{C}/\mathtt{L} \rceil$ chunks of $\mathtt{L}$ bits each, and we look up each chunk individually. This reduces the table size to $2^{\mathtt{L}}$ entries. Let K represent the maximum number of blocks per chunk (also computable from public parameters).

The key challenge is that a block may span multiple consecutive chunks, so we cannot count blocks independently within each chunk. Our solution tracks what color each chunk boundary has, allowing us to correctly identify and merge blocks that span chunk boundaries. Then, at the end, we will check the consistency of these block chunks with a prover-provided array of single block

lengths.

Block Measurement Lookup Tables

Table 1: Chunk Encoding (size $2^{\mathtt{L}}$)
Each entry corresponds to a possible $\mathtt{L}$-bit chunk and contains a tuple $(c, \tilde{c}, r, nb, \text{odd}, \text{black})$ such that:

- $c \in [2^{\mathtt{L}}]$: The chunk encoded as a field element (interpreting the chunk bit string as a binary number).
- $\tilde{c} \in [B^{\mathtt{L}-1}]$: An encoding of the block lengths within this chunk that cannot overlap with the next chunk, represented as a base-B number.
- $r \in [B]$: An encoding of the block that might overlap with the next chunk in base-B.
- $nb \in [\mathtt{L}]$: The number of blocks in the chunk (not including r).
- odd $\in \{0, 1\}$: $nb \mod 2 == 1$
- black $\in \{0, 1\}$: Set to 1 if the chunk represented by r that may overlap is black.

This table has length $2^{\mathtt{L}}$, which is every possible value of the length-$\mathtt{L}$ chunk.
Table 2: Powers of B (size K)
Each entry provides (i, B^i), that is, an exponent and the public parameter B raised to the ith power.
This table has length K, which is the upper bound of the number of blocks in a chunk.
Table 3: Range check
An array of values $[0, \mathtt{R} - 1]$ for range checks.

Figure 5: Block measurement lookup tables

5.2.1 Processing chunks

We process chunks one-by-one and pack block lengths into base-B encoded values that break at block boundaries instead of the arbitrary chunk boundaries. If chunk i ends with a black block and chunk $i + 1$ starts with a black block, these blocks are part of the same run and must be merged (similarly for white and white blocks). We also keep track of the number of blocks in each chunk as we go. See Algorithm 4. This process makes use of several lookup tables – see Figure 5.

5.2.2 Consistency check

From the previous part we have an array c' of encoded chunks divided on block boundaries, and we have an array n' such that $n'[i]$ is the number of blocked encoded in $c'[i]$. Let the prover provide b, an array of single block lengths. We need to show consistency between c', n' and b. We will do this with a randomized polynomial identity check similar to the technique described in Section 4.1.

We need to unpack the elements of c' and show they are consistent with b. First, the prover demonstrates a base-B decomposition of each encoded chunk, unpacking it. Let $c'[i][j]$ represent the jth element of the unpacking of $c'[i]$.

Algorithm 4 Process chunks

$c \leftarrow r/\texttt{L}$ ▷ partition row in to $m = \lceil \texttt{C}/\texttt{L} \rceil$ chunks
$r \leftarrow 0, \text{black}_r \leftarrow 0$ ▷ initialize accumulators
$c' \leftarrow [0; m], n' \leftarrow [0; m]$ ▷ initialize output arrays
for $i \leftarrow 0$ **to** $m - 1$ **do**
  $(\tilde{c}_i, r_i, nb_i, \text{odd}_i, \text{black}_i) \leftarrow \text{lookup}(c[i], \text{Table1})$
  $e \leftarrow \text{black}_i \oplus \text{black}_r \oplus \text{odd}_i$ ▷ compute offset for r
  $B^e \leftarrow \text{lookup}(e, \text{Table2})$
  $c'[i] \leftarrow r \cdot B^e + \tilde{c}_i$ ▷ add r onto $\tilde{c}$
  $n'[i] \leftarrow nb_i + e$
  $r \leftarrow r_i$, $\text{black}_r \leftarrow \text{black}_i$ ▷ set accumulators
end for

The prover constructs the following two polynomials:

$$f(x) = \sum_{i=0}^{B} b[i] \cdot x^i \qquad\qquad g(x) = \sum_{i=0}^{m} (\sum_{j=0}^{K} c'[i][j] \cdot x^j) x^{\sum_{k=0}^{i-1} n'[k]}$$

We encode $f(\beta) == g(\beta)$ in I-R1CS for random verifier challenge β. By the Schwartz-Zippel Lemma, this shows that $f(x) = g(x)$ (except with negligible soundness error), and the measurements extracted from the image are consistent with the block measurements the prover provided.

5.2.3 Splitting the image into "well-behaved" and "garbage" rows

Without any assumptions on the image structure, the upper bounds for block length, number of blocks, and blocks per chunk are, in the worst case, $B = \texttt{C} + 1$, $M = \texttt{C}$, and $K = \texttt{L}$ respectively. However, for rows that follow PDF417 structure (which we call *well-behaved*), we can get tighter bounds on these parameters. Using lower B and K lowers the constraint count.

Even for a valid scan of a PDF417 barcode, it is not necessarily the case that all rows of pixels are well-behaved – the image may have rows of pixels that are on the boundaries between logical rows in the barcode as a consequence of overlaying a grid of pixels onto the logical rows and columns of the barcode. We call these rows that are not well-behaved *garbage* rows. (These are always full rows in our setting because we assume the barcode and has no rotation or skew—Section 3.2.2.)

We allow the prover to specify which rows are well-behaved and which are garbage. Let $\texttt{NR}$ be the maximum number of logical rows in a PDF417 barcode; there are at most $\texttt{NR} - 1$ garbage rows positioned in between these logical rows. We will later verify that the division is legitimate to ensure a cheating prover has not tried to throw out good data (Section 5.4). For now, we just ensure that every row of the image has been specified as either well-behaved or garbage, and no rows have been added or skipped.

Let $P[\texttt{R}][\texttt{C}]$ denote the whole image, $G[\texttt{G\_R}][\texttt{C}]$ denote the garbage rows, and $W[\texttt{R}][\texttt{C}]$ denote the well-behaved rows. We need to show that P has been appropriately split into G and W. Note that W is the same size as P as to not reveal how many rows were removed as garbage. G can be smaller since the maximum number of logical barcode rows is specified in the PDF417 spec and known to the verifier. Let the prover also provide $g[\texttt{G\_R}]$ and $w[\texttt{R}]$ providing the row indexes in P for each of the garbage rows and well-behaved rows respectively. We can show consistency between P, G, and W using this set relation:

$$\{\forall i \in [\texttt{R}],\ (i, P[i])\} \cup \{(-1, \mathbf{0})\} = \{\forall j \in [\texttt{G\_R}], (g[j], G[j])\} \cup \{\forall k \in [\texttt{R}], (w[k], W[k])\}$$

This shows that all rows of P are represented in either G or W, and rows of G, W that are not rows of P must be all zeros. We will use all zeros for extra rows in G, but all zeros are not "well-behaved" according the PDF417 spec, so W will repeat well-behaved rows of P when extras are needed. We also need to show that w is non-decreasing to demonstrate the row order has been maintained in W (it is not important that row order is maintained in G).

To show w is non-decreasing, we use a lookup check in Table 3 (Figure 5); show $w[i] - w[i-1] \in [0, \texttt{R} - 1]$. For the set relation, let the prover provide `row_counts`[R] indicating how many times row i in P is represented in either G or W and `num_zero_rows` indicating how many zero row are in G. We show the set relation using a Habock argument [24] with two verifier challenges α, β:

$$\frac{\texttt{num\_zero\_rows}}{\alpha - 1} + \sum_i \frac{\texttt{row\_counts}[\texttt{R}][\texttt{i}]}{\alpha + i + \beta P[i]} = \sum_j \frac{1}{\alpha + g[j] + \beta G[j]} + \sum_k \frac{1}{\alpha + w[k] + \beta W[k]}$$

Note this is a different use of a Habock argument than usual; instead of a standard lookup into a public table, we use it to check the above more general set relation.

In order to do this, we need to represent each row of binarized pixels as a single field element. We can do that naively with another verifier challenge γ as $A[r] = \sum_{c=0}^{\texttt{C}} A[r][c] \cdot \gamma^c$ - this requires C constraints. To do this more efficiently, we can group many pixels together as one field element represented in binary (sound as long as we don't overflow the field modulus). Then we use these values in the randomized sum.

5.3 Normalization

Starting from the lengths of blocks measured in pixels from Block Measurement (Section 5.2), the goal of normalization is to convert those measurements from *pixels* to *modules*, and then group the modules into *words*, or collections of eight blocks. We then represent each word as one field element by encoding it base-18. A word can be at most 17 modules long, thus a block must definitely be no more than 17 modules long, so base-18 ensures no module measurement overflows. Details are in Appendix C.2.

5.4 Codeword interpretation

The goal of codeword interpretation is to retrieve the encoded PDF417 codewords given normalized block measurements (or "words" - see Section 5.3) for both the well-behaved and garbage portions of the image (see Section 5.2.3). Additionally, show that the parts of the barcode the prover indicates should be discarded are legitimately not useful to decoding. Note this is an additional requirement over the base algorithm that is necessary for soundness. Without this check, the prover could throw out subsets of the barcode data by marking a well-behaved row as garbage.

If logical rows lined up exactly with the image's "physical" rows of pixels, codeword interpretation would be a simple lookup table converting words to PDF417 codewords. However, since this is not the case, there are words that are either (1) duplicates of the word above them that should not be recorded twice, or (2) "garbage" in between logical rows that may not decode to a codeword at all. The circuit needs to discard these extra words before doing the lookup, but it needs to make sure that the data it discards is legitimate and not data that should be decoded.

Difficulty arises because well-behaved rows can have image artifacts or scanning errors that cause decoding failures and make them plausibly garbage rows. Likewise, garbage can by chance successfully decode and look like a well-behaved row. The goal is not to cryptographically verify that the barcode decodes to the intended data; the PDF417 spec does not support this kind of guarantee. Instead, we only want to verify that the prover has executed the decoding algorithm correctly. Also

recall that there are multiple "correct" implementations of PDF417 decoding (Section 2.4). The goal here is to ensure that the prover has done something within the bounds of the spec instead of verifying the prover has arrived at some canonical answer.

To ensure the prover has labeled some correct set of rows for decoding and appropriately discarded the rest, we track the logical row number indicated by the left row indicator. As we iterate through the rows, the left row indicator should decode to incrementing row numbers for all rows the prover selects to decode. This ensures that the prover has not dropped or added garbage rows in the middle of the barcode. Note that it is possible that a "garbage" row happens to have a left row indicator decoding to the correct row number, and the prover could throw out a well-behaved row and exchange it for that one. However, this is a valid decoding choice; the left row indicator shows the garbage row is in that logical row, so that "garbage" row is not truly garbage after all.

This tracking still allows the prover to drop rows from the end of the barcode, or possibly add more rows at the end of the barcode if the left row indicators of those rows happen to decode as such. This is handled in Section 5.7.

Within a row that the prover decides to decode, there may still be words that do not decode due to scanning errors. We verify this is a legitimate erasure by showing that word is disjoint with the set of words that decode using a proof of set disjointness based on Bézout's identity from [23].

Codeword Interpretation

1. **Check dummies** - The prover provides `words_with_dummies[R][WB_W]`: an identical array to `words[R][WB_W]`, except that unneeded values are replaced with zeros. Compare `words` to `words_with_dummies` and check that all the elements of `words_with_dummies` are either zero or equal to the corresponding element of `words`.

2. **Accumulate codewords** - Iterate row-by-row through each word. Ensure the left row indicator decodes to the appropriate row number. If the word is zero'd out in `words_with_dummies`, add it to `garbage`.

3. **Verify the marked garbage**. Use a set disjointness check to ensure that elements of `garbage` and at least one element per row of `garbage_words[R][G_W]` should not decode in the codewords table.

4. **Verify the prover-provided `codewords[W]` in the augmented witness is consistent with the output of Step 2.** Use the technique described in Section 4.1 to compare `codewords[W]` and `cw_and_dummies[R][WB_CW]`.

5. **Verify the prover-provided `left_row_inds[NR]` is consistent with the output of Step 2** in the same manner as Step 4.

Figure 6: Codeword interpretation algorithm

5.5 Barcode metadata

The goal of this step is to compute the number of rows, number of columns, and error correction level for the barcode (all considered part of the secret data) based on the barcode's left row indicators. This is done by checking a system of equations specified by the PDF417 spec. Then, based on

error correction level, compute the number of error correction codewords the barcode should have. Details can be found in Appendix C.3.

5.6 Error correction

The goal of this step is to check that BCH Reed-Solomon error correction has been performed correctly over the codewords extracted from the barcode image. Let the prover provide the corrected codewords in the augmented witness and check correctness instead of performing the whole algorithm in-circuit.

To check BCH RS error correction, it suffices to check:

1. the error polynomial is sufficiently sparse, and
2. the corrected codeword polynomial evaluates to zero at all relevant evaluation points.

Details can be found in Appendix C.4.

5.7 Barcode size

The goal of this step is to check that the number of codewords extracted from the barcode is consistent with metadata found in the left row indicators (Section 5.5) and the symbol length descriptor codeword (Section 2.4).

As discussed in Section 5.4, a discrepancy might occur if (1) scanning errors slightly mangled the decoded message, and (2) a cheating prover is maliciously attempting to add or remove rows at the end of the barcode.

To ensure soundness, we check that the total number of codewords and the number of data codewords the prover provided in codeword interpretation is consistent with the barcode metadata extracted from left row indicators in Section 5.5 to ensure we have arrived at the correct amount of data. Details can be found in Appendix C.5.

5.8 Character interpretation

In character interpretation, corrected codewords are converted into the ASCII character data they encode. In PDF417 barcodes, this requires using three state machines. Through our assumption of text mode (Section 3.2.4) we omit one and are left with the following two state machines:

- An outer state machine tracking which portion of the codeword array we are in - error correction codewords, pad codewords, data codewords, or leading zeros (necessary to make the codewords array constant length).
- An inner state machine tracking which text mode *sub-mode* we are in - alpha, lower, mixed, and punctuation. Each sub-mode has its own lookup table mapping base-30 values to either an ASCII character or an indication to change to a different sub-mode. *Note the data-dependence: the lookup table utilized at any given time depends on the previously-decoded symbols.*

The outer state machine determines where we are in the `corrected_cw[W]` array and specifies whether or not interpretation should be done on this particular codeword. If the codeword should be interpreted, we use a lookup table that encodes all valid transitions to check the inner state machine as described in Section 4.2 (see Algorithm 5). Lastly, the prover demonstrates that it has correctly stripped leading, following, and interleaved null characters using the technique described in Section 4.1.

Algorithm 5 Character interpretation

```
corrected_cw[W]                                          ▷ corrected codewords
ts[2W]                                                   ▷ prover-provided table states
chars[2W] ← [0; 2W]                                      ▷ initialize output array
for i ← 0 to 2W − 1 do
    lookup((ts[2i − 1], ts[2i]), TRANSITIONS)
    lookup((ts[2i], ts[2i + 1]), TRANSITIONS)
    if i ∈ [num_ec_cw] then
        ▷ in error correction codewords
        ts[2i], ts[2i + 1] == EC_STATE
        chars[2i], chars[2i + 1] ← 0
    else if corrected_cw[i] == 900 then
        ▷ pad codeword
        ts[2i], ts[2i + 1] == PAD_STATE
        chars[2i], chars[2i + 1] ← 0
    else if ts[2i], ts[2i + 1] == SLD_STATE then
        ▷ lookup table ensures prover can't move into this state when it shouldn't
        chars[2i], chars[2i + 1] ← 0
    else
        corrected_cw[i] == 30ts[2i].char
                  +ts[2i + 1].char
        chars[2i] ← ts[2i].char
        chars[2i + 1] ← ts[2i + 1].char
    end if
end for
```

5.9 Proving consistency of zkSNARK with C2PA

The root of trust for our system is the C2PA-enabled camera, which asserts image validity with a signature over the image and metadata. According to the C2PA specification [14], the image and metadata should be hashed using SHA2, and then the hash is signed. As noted by other ZK image processing works [17,18], proving a valid pre-image for a SHA2 hash in zero knowledge for an input as large as an image is infeasible. Therefore, we use a technique similar to [17], but modified for the Spartan proof system. The camera signs a polynomial commitment instead of a SHA2 hash (see Recommendation 2 in Section 8), and then we modify the Spartan proof system entry slightly to show consistency between that commitment and the polynomial commitment inherent in the first step of the Spartan proof system. The camera generates its own polynomial commitment, and then we change Spartan to utilize this commitment unmodified when committing to those witness inputs generated directly by the camera (the image). Since the polynomial commitment generated by the camera is hiding and randomized, this protocol is still zero-knowledge.

We can take in the camera-generated polynomial commitment to the image alongside a separate polynomial commitment to the rest of the witness generated by the prover using the optimization presented in the Spartan paper discussed in Section 2.3.3. Call the image I and the other private inputs p, and call the public inputs x. Using the same strategy as Section 2.3.3, the prover can present two polynomial commitments, one for each of $\tilde{I}$ and $\tilde{p}$, and the verifier can reconstruct $\tilde{Z}$.

Assume vector z is constructed as $I||p||1||x$. Assume for simplicity that $|I| = |p| = 0.5(|x| + 1)$.

Then, $\tilde{Z}$ can be computed by the verifier as:

$$\begin{aligned}\tilde{Z}(r) =&(1-r[0])(1-r[1])\cdot\tilde{I}(r[2...])\\&+(1-r[0])\cdot r[1]\cdot\tilde{p}(r[2...])\\&+r[0]\cdot\tilde{(x,1)}(r[1...])\end{aligned}$$

The lengths can be adjusted as needed, as long as each segment is a power of two in length, and the total vector is also a power of two. For example, say $|I| = 2^i$, $|p| = 4 \cdot |I| = 2^{i+2}$, $|x| + 1 = |p| = 2^{i+2}$. Then, we use four selector bits to compute $\tilde{Z}$ as follows:

$$\begin{aligned}\tilde{Z}(r) =&(1-r[0])(1-r[1])(1-r[2])(1-r[3])\cdot\tilde{I}(r[4...])\\&+r[0]\cdot(1-r[1])\cdot\tilde{p}(r[2...])\\&+r[0]\cdot r[1]\tilde{(x,1)}(r[2...])\end{aligned}$$

Note that this requires that the camera pad the image to a power of two elements before signing.

6 Experimental Results

6.1 Implementation details

We implemented our constraint system using ZoKrates [31] compiled with CirC [30]. We made some changes to the CirC compiler to support tuple-valued lookups and compilation of large circuits, which we plan to integrate into the main CirC compiler. We used the Dorian proof system for our backend to get the fast prover times present in Spartan and allow R1CS with random challenge [1]. We modified Dorian to allow for the external commitment input described in Section 5.9. We use rxing [29] to decode PDF417 barcodes, and we modified it to generate the auxiliary input needed for our prover. All code for this project, including modifications to existing tools, is open source[1].

All measurements were run on a Linux server with an AMD EPYC 9124 processor, 32 CPU cores, and 604GB of RAM. All reported numbers are averages across five runs. We were limited in computation resources for CirC compilation, thus limiting the circuit size that we could measure. Further performance engineering work could make these benchmarks possible. For comparisons with a zkVM, we used the SP1 zkVM [32] run on a minimally-modified version of rxing [29] (the only modification was to remove a timing library not supported by SP1). We chose SP1 because it supports the Rust standard library, and it required minimal engineering to get it working with existing decoding tools.

6.2 Benchmarks

Figure 7 shows how our system compares to the existing alternative, a zkVM. Across all problem sizes, our circuit is at least **18.6x faster** and uses **6.9x less RAM**. Also notably our prover times are roughly comparable to other ZK image processing work at these image sizes [17], even though we consider a much more complex computation. See Table 1 for prover time, verifier time, and prover memory benchmarks.

Figure 8 shows how the individual sub-circuits compare to each other when total time is normalized across parameter regimes. Note that `words_pipeline` is the combination of normalization (Section 5.3) and codeword interpretation (Section 5.4) because the output of normalization is

[1]Code found at `https://github.com/sschefflab/zk-barcodes`

Table 1: Full circuit benchmark results (mean over 5 runs)

Image (px)	Barcode (px)	Max barcode (logical)	Max EC level	Chunk size	Prover time (s)	Prover RAM (GB)	Verifier time (s)
144x48	120x48	3×3	0	10	0.99	0.68	0.14
144x48	120x48	3×3	1	10	0.98	0.69	0.14
192x144	120x144	3×3	0	10	1.90	1.51	0.23
192x144	120x144	3×3	1	10	1.93	1.52	0.23
192x144	171x140	10×6	0	9	2.74	2.36	0.37
192x144	171x140	10×6	1	9	2.75	2.37	0.37
384x288	360x288	3×3	0	10	6.15	5.42	0.75
384x288	360x288	3×3	1	10	6.23	5.45	0.77
384x288	342x280	10×6	0	9	6.44	5.89	0.80
384x288	342x280	10×6	1	9	6.50	5.92	0.80
384x288	290x273	21×13	0	10	7.04	6.53	0.84
384x288	290x273	21×13	1	10	7.05	6.53	0.84
384x288	290x273	21×13	5	10	6.96	6.53	0.85
384x288	290x273	90×30	8	10	13.55	15.11	2.07

needed as an input to codeword interpretation, and there is not a way to run "verify" in between those circuits without additional witness elements. As expected, `binarize`, `block_measurement`, and `words_pipeline` dominate, as those circuits depend on the size of the barcode in pixels. Beyond that point, the circuits only depend on logical barcode parameters like the number of codewords or error correction level, and these parameters are all much smaller than the image size.

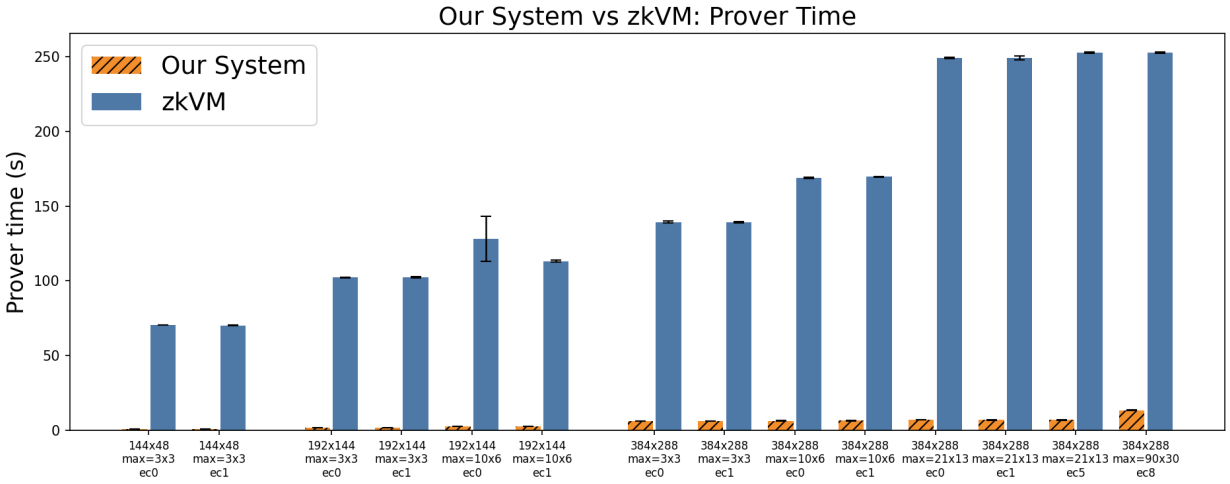
7 Security proof sketches

This section gives short explanations for how our system meets our security goals presented in Section 3.1.

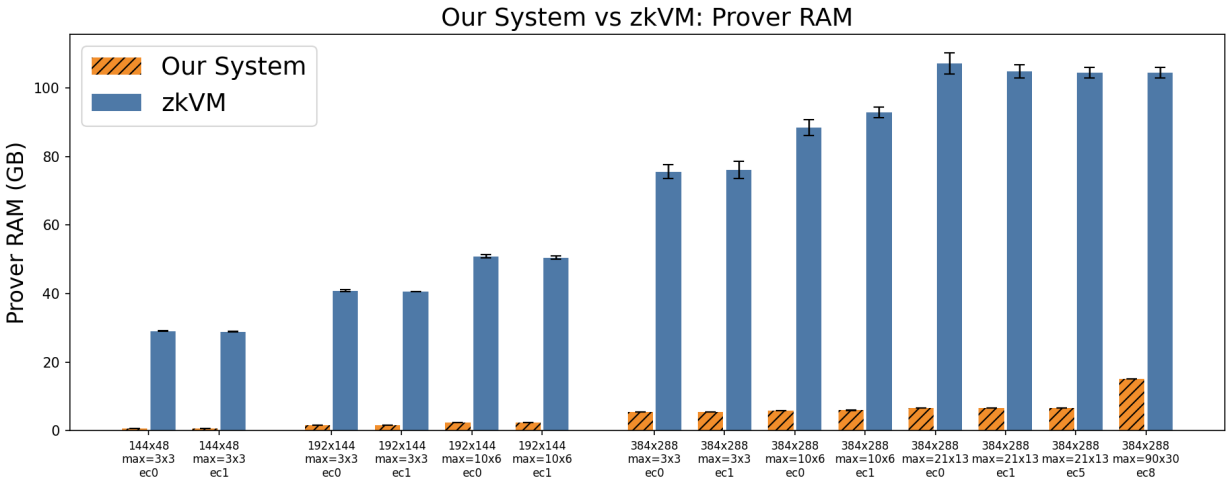
1. **Completeness** (Definition B.1) follows from Dorian's completeness property, the correctness of the signature scheme, and the correctness of the polynomial commitment scheme. (As noted in Section 3.2, we exclude a small subset of PDF417 barcodes.)

2. **Soundness against inauthentic images** (Definition B.2) follows from the signature scheme's unforgeability property and the binding property of the polynomial commitment scheme. An adversary cannot forge a signature on a digest of its choosing by unforgeability, and it cannot substitute a different image for a signed digest by the binding property.

3. **Knowledge soundness** (Definition B.3) follows from the binding property of the polynomial commitment scheme and Dorian's knowledge soundness property. Dorian's knowledge soundness property means that the prover must know some image used in the Dorian proof, and since we use the digest generated by the camera as a direct input to Dorian, we get consistency with the digest "for free." The binding property of the PCS then ensures that an adversary cannot substitute another image of its choosing as a pre-image for the camera-generated digest.



(a) Prover runtime of our system vs zkVM



(b) Prover memory usage of our system vs zkVM

Figure 7: Prover performance of our system vs zkVM

4. **Zero knowledge** (Definition B.4) follows from the hiding property of the polynomial commitment scheme and Dorian's zero-knowledge property. However, as noted in Section 3.2.1, since the coordinates of the barcode in the image are revealed in our circuit, this could reveal sensitive information in some applications.

8 Recommended changes to C2PA

We make the following recommendations to the C2PA standard to better support zero-knowledge image processing compatible with proofs of authenticity:

Recommendation 1: ZK-friendly hashing algorithms. As noted in Section 5.9, C2PA's requirement that SHA2 be used as the hash makes authenticated zero-knowledge image processing too slow. We recommend the C2PA standard be updated to include more zero-knowledge-friendly hash functions like Poseidon [33] (also used by [18]).

Recommendation 2: Other commitment types. Along with ZK-friendly hashes, we

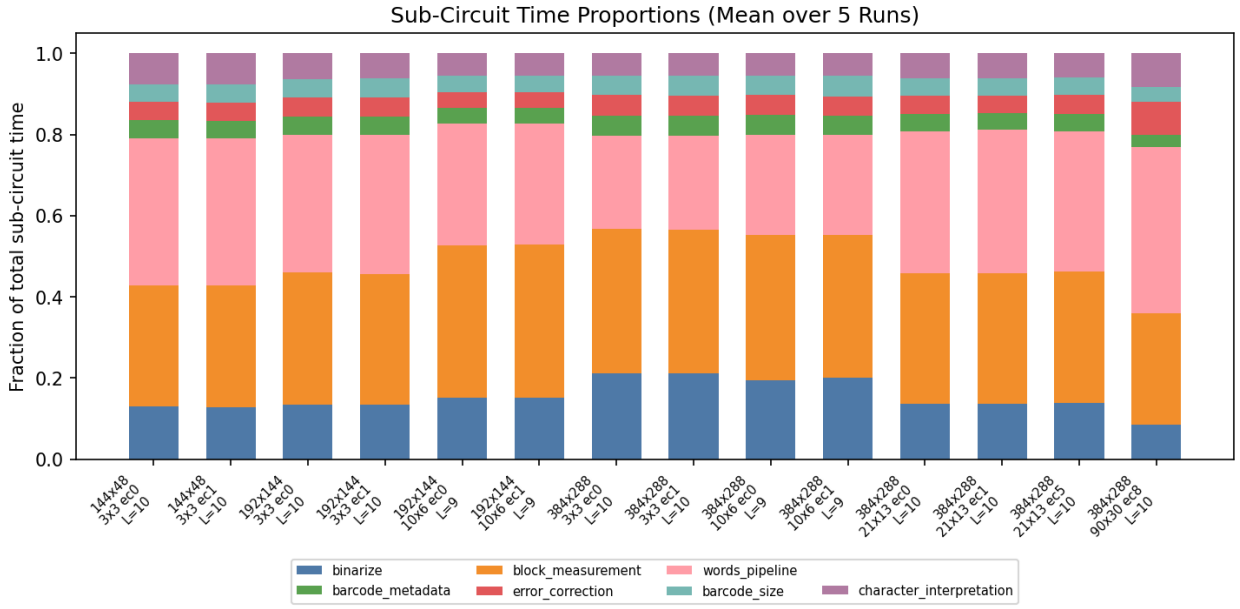


Figure 8: Sub-circuit times, normalized

recommend the C2PA standard be updated to include other commitment types like polynomial commitment schemes. That would allow for the use of the techniques presented in this work and in [17]. Many such schemes require inputs to be of particular length; in this work, we require a power of two. Thus, we also recommend C2PA allow for padding the image before committing to it.

Recommendation 3: Ring signatures. To avoid revealing which specific device took the photo, C2PA should allow ring signatures over a set of device keys—even the whole registry—so images can be attested as authentic without exposing the signer's identity. This could be scoped to a brand or model, e.g., "a Google Pixel 15" or "The New York Times," enabling social-media photos to be checked for authenticity without mandating a full individual-level PKI.

This kind of privacy guarantee could also be helpful in journalistic settings; a journalist may want to prove that an image of some sensitive event is real without the image being traceable to a specific camera or individual.

Recommendation 4: Sign binarized images. One immediate way to remove about 10-20% of our prover runtime would be to have the signed image already be binarized. This could be handled easily on the camera side with expanded smart phone camera APIs rather than adding an additional step to prove in ZK.

9 Related Work

We discuss prior work on image processing under zero knowledge and privacy-preserving ID-based credential verification. See Table 2 for a summary.

Zero-knowledge image transformations. Starting from a signed image, this line of work aims to prove the correctness of subsequent transformations.

PhotoProof [34] introduced the idea of proving image transformations on signed images in the paradigm of proof-carrying data. Ko et al. [35] prove correctness of image redaction on signed images and replace the hash with a Pedersen commitment. ZK-IMG [36] uses Halo2 to prove

Table 2: Comparison with prior systems

Work	Authenticated image as input?	Data-dependent image processing?	Proves possession of credential?	Proves image → credential?
PhotoProof (2016) [34]	✔	generic	✖	✖
Ko et al. (2021) [35]	✔	✖	✖	✖
ZK-IMG (2022) [36]	✔	✖	✖	✖
VerITAS (2025) [17]	✔	✖	✖	✖
VIMz (2025) [18]	✔	✖	✖	✖
Trust Nobody (2025) [37]	✔	✖	✖	✖
zk-REAL (2026) [19]	✔	✖	✖	✖
zk-creds (2023) [10]	✖	N/A	✔	✖
Longfellow (2024) [11]	✖	N/A	✔	✖
Crescent (2024) [38]	✖	N/A	✔	✖
Vega (2025) [39]	✖	N/A	✔	✖
Anon Aadhaar (2023-5) [40]	✖	N/A	✔	✖
ZKPassport (2023-6) [41]	✖	N/A	✔	✖
Self (2023-6) [42]	✖	N/A	✔	✖
This work	✔	✔	✔	✔

correctness of several image transformations and shows that hash verification dominates prover time. VerITAS [17] overcomes this bottleneck by replacing the hash with a hash involving lattices and Poseidon [33] for four data-independent transformations; it also presents an option using a polynomial commitment scheme, an approach that we also adopt in this work. Zk-REAL [19] similarly uses a lattice-based hash, achieving roughly 29% faster proving than ZK-IMG. VIMz [18] substitutes Poseidon for SHA-256 and boosts efficiency through a folding-based proof, exploiting the fact that their transformations act independently on image rows. Trust Nobody [37] applies an analogous tiling approach but retains SHA-256, limiting its performance.

Dealing with the hash: ZK-IMG and Trust Nobody retain the SHA-256 hash from C2PA, their ZK performance suffers accordingly. VerITAS and zk-REAL adopt lattice-based hashes to mitigate this. Our system instead has the C2PA signer commit to the image data as in VerITAS, yielding a substantial improvement in prover time.

Data-dependent computation: All prior image transformations investigated under zero knowledge—crop, greyscale, resize, blur, contrast, censoring, rotation, and similar operations—apply a fixed algorithm with no conditional branching on secret pixel values. Our work is *the first to implement highly data-dependent image processing algorithms under zero-knowledge.*

Non-local secret input processing: Prior work also benefits from transformation locality: VIMz's folding scheme and Trust Nobody's tiling approach work precisely because each operation touches only nearby pixels. Our barcode-decoding algorithm must process the entire image with many branches conditioned on secret data, so these efficiency gains are unavailable to us. We hope that this work serves as a stepping stone toward more work on image processing techniques under zero knowledge that require conditional data-dependency and non-local processing of pixels.

Zero-knowledge attribute proofs from previously-issued credentials. A maturing line of work aims to allow individuals to prove that they have some attribute (e.g. age over 18) using an externally-issued ID, but without relying on the ID issuer to support the proof itself (beyond a digital signature and possible maintenance of a list of non-revoked credentials). In this vein the closest relative to our work is Dorian [1] which is used to compute proofs of possession for legacy signatures including RSA, ECDSA and EdDSA; our system uses the Dorian zkSNARK as a starting

point.

In the academic literature, zk-creds [10] is the first of a line of modern works to prove attributes and possession of a legacy identity credential signed by an issuing authority, without involving the issuer itself aside from the maintenance of a list of valid credentials. This work was followed by the Groth-based Crescent [38], the Spartan/Nova-based Vega [39], the Ligero-based Longfellow [11], and Dorian [1] as discussed earlier. Many of these works descend from Cinderella [43], a 2016 system for validating X.509 certificates in zero knowledge. An SoK about anonymous credentials for digital identity wallets was recently completed by Bormann and Lehmann in 2026 [44].

Several ZKP-based credential systems are actively deployed in June 2026 or are planned for deployment in late 2026. Based on Frigo and Shelat's Ligero-based anonymous credentials for ECDSA [11], Google has deployed the Longfellow ZK protocol in Google Wallet [45]. The system is currently live for age and ID verification for customers of one European bank, with anticipated widespread expansion within months [46]. EU age-verification efforts [47, 48] include ZKPs in the EU Age Verification Blueprint and accompanying open-source implementations. ZKPassport [41] proves passport attributes from the embedded chip in zero knowledge, AnonAadhaar [40] proves a government signature on attributes, and Self [42] is a commercial wallet supporting private checks on passports and national IDs.

All of these systems operate on digital credentials (bitstrings), so they must address liveness and stolen-credential threats through other means—chip proximity, biometrics, or similar mechanisms, often not integrated into the proof itself. In contrast, our method uses a newly-captured photograph of the ID barcode: it requires no specialized chip, avoids biometric processing, and integrates the image-integrity check directly into the proof.

10 Conclusion and Future Work

We present a method for reading PDF417 barcodes from images in zero knowledge for the purpose of private online attribute verification. Using C2PA-enabled cameras as the root of trust, we build on the proof system Dorian [1] to build our proof system for this highly data-dependent algorithm. Our system is more than 18x faster and requires 6.9x less RAM than the next-best existing alternative. We also contribute techniques of independent interest including new applications of lookup tables and prover advice for data-dependent algorithms. Additionally, we propose changes to the C2PA standard to better support zero-knowledge image processing. Directions for future work include relaxing the assumptions listed in Section 3.2 and benchmarking at larger image sizes to better understand scaling behavior. More broadly, barcodes bridge the physical and digital worlds across many domains and can enable more applications beyond online credential verification. We see this work on private, verifiable barcode decoding as the first step toward more kinds of data-dependent zero-knowledge processing.

References

[1] A. P. Y. Woo, A. Ozdemir, C. Sharp, T. Pornin, and P. Grubbs, " Efficient Proofs of Possession for Legacy Signatures ," in *2025 IEEE Symposium on Security and Privacy (SP)*. Los Alamitos, CA, USA: IEEE Computer Society, May 2025, pp. 3291–3308. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/SP61157.2025.00080

[2] S. T. V. Setty, "Spartan: Efficient and general-purpose zkSNARKs without trusted setup," in *Advances in Cryptology – CRYPTO 2020, Part III*, ser. Lecture Notes

in Computer Science, vol. 12172. Springer, 2020, pp. 704–737. [Online]. Available: https://doi.org/10.1007/978-3-030-56877-1_25

[3] Free Speech Coalition. State age verification laws. [Online]. Available: https://action.freespeechcoalition.com/age-verification-resources/state-avs-laws/

[4] U. Kingdom, "Online safety act 2023," 2023. [Online]. Available: https://www.legislation.gov.uk/ukpga/2023/50/contents/enacted

[5] European Commission, 4 2026, https://digital-strategy.ec.europa.eu/en/policies/eu-age-verification.

[6] S. D. H. B. 3424, 2024. [Online]. Available: https://sdlegislature.gov/Session/Bill/25525/282508

[7] S. Liu and S. Scheffler, "Adequately tailoring age verification regulations," in *Proceedings of the Symposium on Computer Science and Law*, ser. CSLAW '26. New York, NY, USA: Association for Computing Machinery, 2026, p. 239–248. [Online]. Available: https://doi.org/10.1145/3788646.3789526

[8] C. D. Hylender, P. Langlois, A. Pinto, and S. Widup, "2025 data breach investigations report," Verizon, Technical Report, 2025, 18th annual edition. [Online]. Available: https://www.verizon.com/business/resources/reports/dbir/

[9] Bright Defense. 120 data breach statistics for 2025. Bright Defense. [Online]. Available: https://www.brightdefense.com/resources/data-breach-statistics/

[10] M. Rosenberg, J. D. White, C. Garman, and I. Miers, "zk-creds: Flexible anonymous credentials from zkSNARKs and existing identity infrastructure," in *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2023, pp. 790–808. [Online]. Available: https://doi.org/10.1109/SP46215.2023.10179430

[11] M. Frigo and abhi shelat, "Anonymous credentials from ECDSA," Cryptology ePrint Archive, Paper 2024/2010, 2024. [Online]. Available: https://eprint.iacr.org/2024/2010

[12] ——, "The Longfellow Zero-knowledge Scheme," Internet Engineering Task Force, Internet-Draft draft-google-cfrg-libzk-01, Sep. 2025, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/draft-google-cfrg-libzk/01/

[13] S. Liu and S. Scheffler, "Privacy-preserving age verification based on improved verifiable credentials framework," in *Workshop on Technology and Consumer Protection (ConPro)*. IEEE, 2025. [Online]. Available: https://conpro25.ieee-security.org/papers/liu-conpro25.pdf

[14] Coalition for Content Provenance and Authenticity, "Content credentials: C2PA technical specification," May 2025, version 2.2, 2025-05-01. [Online]. Available: https://spec.c2pa.org/specifications/specifications/2.2/index.html

[15] C2PA Technical Working Group Conformance Task Force, "C2pa generator product security requirements," 6 2025. [Online]. Available: https://github.com/c2pa-org/conformance-public/blob/main/docs/current/C2PA%20Generator%20Product%20Security%20Requirements.pdf

[16] I. Reynolds. (2025, 8). [Online]. Available: https://blog.google/products/pixel/pixel-10-camera-features/

[17] T. Datta, B. Chen, and D. Boneh, " VerITAS: Verifying Image Transformations at Scale ," in *2025 IEEE Symposium on Security and Privacy (SP)*. Los Alamitos, CA, USA: IEEE Computer Society, May 2025, pp. 4606–4623. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/SP61157.2025.00097

[18] S. Dziembowski, S. Ebrahimi, and P. Hassanizadeh, "Vimz: Private proof sof image manipulation using folding-based zksnarks," in *2025 Proceedings on Privacy Enghancing Technologies Symposium*. Privacy Enhancing Technologies Board, 2025, pp. 344–362. [Online]. Available: https://doi.org/10.56553/popets-2025-0065

[19] A. Koyama, K. Toyoda, M. Fujimoto, and T. H. Tran, "zk-real: A zero-kknowledge-based protocol for repeated image edit authenticity proof with lattice hashing," *IEEE Open Journal of the Computer Society*, vol. 7, pp. 443–454, 2026. [Online]. Available: https://doi.org/10.1109/OJCS.2026.3661052

[20] J. Katz and Y. Lindell, *Introduction to Modern Cryptography*, 2nd ed., ser. Cryptography and Network Security. Chapman & Hall/CRC, 2014.

[21] B. Barak and Y. Lindell, "Strict polynomial-time in simulation and extraction," in *Proceedings of the Thiry-Fourth Annual ACM Symposium on Theory of Computing*, ser. STOC '02. New York, NY, USA: Association for Computing Machinery, 2002, p. 484–493. [Online]. Available: https://doi.org/10.1145/509907.509979

[22] A. Kate, G. M. Zaverucha, and I. Goldberg, "Constant-size commitments to polynomials and their applications," in *Advances in Cryptology - ASIACRYPT 2010*, M. Abe, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 177–194.

[23] A. Ozdemir, E. Laufer, and D. Boneh, "Volatile and persistent memory for zksnarks via algebraic interactive proofs," in *2025 IEEE Symposium on Security and Privacy (SP)*, 2025, pp. 3309–3327.

[24] U. Haböck, "Multivariate lookups based on logarithmic derivatives," Cryptology ePrint Archive, Paper 2022/1530, 2022. [Online]. Available: https://eprint.iacr.org/2022/1530

[25] "ISO/IEC 15438:2015: Information technology — automatic identification and data capture techniques — pdf417 bar code symbology specification," International Organization for Standardization and International Electrotechnical Commission, Sep. 2021, standard. [Online]. Available: https://www.iso.org/standard/65502.html

[26] "AAMVA DL/ID card design standard: Personal identification – AAMVA north american standard," American Association of Motor Vehicle Administrators, Jun. 2025, standard. [Online]. Available: https://www.aamva.org/getmedia/81af105d-8b1b-45e1-aa46-f1800a259ed1/AAMVADLIDCardDesignStandard2025.pdf

[27] zxing-cpp contributors, "ZXing-C++: An open-source multi-format barcode image processing library," 2026, apache-2.0 License. C++ port of the Java ZXing library, with additional enhancements. [Online]. Available: https://github.com/zxing-cpp/zxing-cpp

[28] Sparkfish LLC, "pdf417decoder: PDF417 decoder available in Python," cPOL License. Pure Python port of a C# library by Uzi Granot. Source: https://github.com/sparkfish/pdf417decoder. [Online]. Available: https://pypi.org/project/pdf417decoder/

[29] H. A. Schimke, "rxing: cRustacean Crossing – a Rust port of the ZXing barcode library," 2026, apache-2.0 License. Rust port of ZXing (https://github.com/zxing/zxing) with enhancements from zxing-cpp. [Online]. Available: https://crates.io/crates/rxing

[30] A. Ozdemir, F. Brown, and R. S. Wahby, "Circ: Compiler infrastructure for proof systems, software verification, and more," in *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2022, pp. 2248–2266. [Online]. Available: https://doi.org/10.1109/SP46214.2022.9833782

[31] "Zokrates," ZoKrates Project, 2026, open-source implementation. [Online]. Available: https://github.com/Zokrates/ZoKrates

[32] Succinct Labs, "SP1: A zero-knowledge virtual machine," https://github.com/succinctlabs/sp1, 2024, accessed: 2026-06-09.

[33] L. Grassi, D. Khovratovich, C. Rechberger, A. Roy, and M. Schofnegger, "Poseidon: A new hash function for Zero-Knowledge proof systems," in *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, Aug. 2021, pp. 519–535. [Online]. Available: https://www.usenix.org/conference/usenixsecurity21/presentation/grassi

[34] A. Naveh and E. Tromer, "Photoproof: Cryptographic image authentication for any set of permissible transformations," in *2016 IEEE Symposium on Security and Privacy (SP)*, 2016, pp. 255–271.

[35] H. Ko, I. Lee, S. Lee, J. Kim, and H. Oh, "Efficient verifiable image redacting based on zk-snarks," in *ASIA CCS '21: ACM Asia Conference on Computer and Communications Security, Virtual Event, Hong Kong, June 7-11, 2021*, J. Cao, M. H. Au, Z. Lin, and M. Yung, Eds. ACM, 2021, pp. 213–226. [Online]. Available: https://doi.org/10.1145/3433210.3453110

[36] D. Kang, T. Hashimoto, I. Stoica, and Y. Sun, "ZK-IMG: attested images via zero-knowledge proofs to fight disinformation," *CoRR*, vol. abs/2211.04775, 2022. [Online]. Available: https://doi.org/10.48550/arXiv.2211.04775

[37] P. D. Monica, I. Visconti, A. Vitaletti, and M. Zecchini, "Trust nobody: Privacy-preserving proofs for edited photos with your laptop," in *2025 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2025, pp. 4624–4642. [Online]. Available: https://doi.org/10.1109/SP61157.2025.00014

[38] C. Paquin, G.-V. Policharla, and G. Zaverucha, "Crescent: Stronger privacy for existing credentials," 2024, preprint. [Online]. Available: https://eprint.iacr.org/2024/2013

[39] D. Kaviani and S. T. V. Setty, "Vega: Low-latency zero-knowledge proofs over existing credentials," Accepted to the 2026 IEEE Symposium on Security and Privacy (SP), 2025, accepted; preprint. [Online]. Available: https://eprint.iacr.org/2025/2094

[40] "Anon aadhaar," Anon Aadhaar, 2026, open-source implementation. [Online]. Available: https://github.com/anon-aadhaar/anon-aadhaar

[41] "ZKPassport circuits," ZKPassport, 2026, open-source implementation. [Online]. Available: https://github.com/zkpassport/circuits

[42] "Self," Self, 2026, open-source implementation. [Online]. Available: https://github.com/selfxyz/self

[43] A. Delignat-Lavaud, C. Fournet, M. Kohlweiss, and B. Parno, "Cinderella: Turning shabby X.509 certificates into elegant anonymous credentials with the magic of verifiable computation," in *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2016, pp. 235–254. [Online]. Available: https://doi.org/10.1109/SP.2016.22

[44] C. Bormann and A. Lehmann, "Sok: Anonymous credentials for digital identity wallets," in *Security Standardisation Research*, H. C. Pöhls and C. J. Mitchell, Eds. Cham: Springer Nature Switzerland, 2026, pp. 3–25.

[45] Google, "Longfellow blog," 8 2025, https://google.github.io/longfellow-zk/blog/.

[46] ——, "Bringing secure digital identity and payment tools to more people," 6 2026, https://blog.google/products-and-platforms/platforms/google-pay/secure-identity-payment-tools/.

[47] "EU age verification blueprint: Architecture and technical specifications," European Commission, 2026, technical specification; working draft. [Online]. Available: https://ageverification.dev/av-doc-technical-specification/docs/architecture-and-technical-specifications/

[48] "Age verification ZKP ICAO for android," European Commission, 2026, open-source implementation. [Online]. Available: https://github.com/eu-digital-identity-wallet/av-lib-android-zkp-age-icao

A Preliminaries

This appendix provides formal definitions for the properties of zero-knowledge proofs and zero-knowledge proofs of knowledge.

Definition A.1 - Completeness. A proof system is *complete* if $\forall(x, w) \in R$:

$$\Pr\left[V(pp, x, \pi) = 1 \middle| \begin{array}{r} pp \leftarrow G(1^\lambda) \\ \pi \leftarrow P(pp, x, w) \end{array}\right] = 1$$

Definition A.2 - Soundness. A proof system is *sound* if for all efficient adversaries P^* and all $(x, w) \notin \mathcal{R}$:

$$\Pr\left[V(pp, x, \pi) = 1 \middle| \begin{array}{r} pp \leftarrow G(1^\lambda) \\ \pi \leftarrow P^*(pp, x, w) \end{array}\right] \leq \text{negl}(\lambda)$$

Definition A.3 - Zero Knowledge. A proof system is *zero-knowledge* if for all efficient adversaries V^*, there exists an efficient algorithm S^{V^*}, such that for all $(x, w) \in \mathcal{R}$ and all efficient distinguisher algorithms D:

$$\Pr\left[D(pp, x, \pi_b) = b \middle| \begin{array}{r} pp \leftarrow G(1^\lambda) \\ \pi_0 \leftarrow P(pp, x, w) \\ \pi_1 \leftarrow S^{V^*}(pp, x) \\ b \xleftarrow{\$} \{0, 1\} \end{array}\right] \leq \frac{1}{2} + \text{negl}(\lambda)$$

Definition A.4 - Knowledge Soundness. A proof system is *knowledge sound* if for all efficient adversaries $P^* = (P_0^*, P_1^*)$, there exists an efficient algorithm ε^{P^*} such that:

$$\Pr\left[V(pp, x, \pi) = 1 \;\middle|\; \begin{array}{r} pp \leftarrow G(1^\lambda) \\ (x, \text{state}) \leftarrow P_0^*(pp) \\ \pi \leftarrow P_1^*(\text{state}) \end{array}\right] -$$
$$\Pr\left[(x, w) \in \mathcal{R} \;\middle|\; \begin{array}{r} pp \leftarrow G(1^\lambda) \\ (x, \text{state}) \leftarrow P_0^*(pp) \\ w \leftarrow \varepsilon^{P_1^*(\text{state})}(pp, x) \end{array}\right] \leq \text{negl}(\lambda)$$

B Formal Security Definitions

This appendix provides formal definitions for our security goals presented in Section 3.1.

Definition B.1 - Completeness. Authentic images with valid decodings should always verify. For all images I:

$$\Pr\left[\begin{array}{l} V(pp, d, \sigma, \\ pk_{cam}, x, \pi) = 1 \end{array} \;\middle|\; \begin{array}{r} (pp, sk_{cam}, pk_{cam}) \leftarrow G(1^\lambda) \\ r \xleftarrow{\$} \mathbb{F} \\ (d, \sigma) \leftarrow \texttt{C2PA.Sign}(sk_{cam}, I, r) \\ (x, aux) \leftarrow \texttt{PDF417.Decode}(I) \\ \pi \leftarrow P(pp, I, x, aux) \end{array}\right] = 1$$

Definition B.2 - Soundness against inauthentic images. Images not captured by the C2PA-enabled camera, instead generated by some efficient malicious algorithm C^*, should be accepted by the verifier with at most negligible probability. For all images I and all (x, aux) such that $\texttt{PDF417.VerifyDecode}(I, x) = 1$:

$$\Pr\left[\begin{array}{l} V(pp, d^*, \sigma^*, \\ pk_{cam}, x, \pi) = 1 \end{array} \;\middle|\; \begin{array}{r} (pp, sk_{cam}, pk_{cam}) \leftarrow G(1^\lambda) \\ r \xleftarrow{\$} \mathbb{F} \\ (d^*, \sigma^*) \leftarrow C^*(I, r) \\ \pi \leftarrow P(pp, I, x, aux) \end{array}\right] \leq \text{negl}(\lambda)$$

Definition B.3 - Knowledge soundness of consistent image. A prover that convinces the verifier must know an image consistent with the digest d containing a barcode that decodes to the data x. More formally, for all adversaries $P^* = (P_0^*, P_1^*)$, if the adversary is able to convince a verifier, there exists an efficient extractor ε that, given access to P^*, can extract image I^* such that digest d opens to I^* and I^* decodes to data x, except with at most negligible probability.

$$\Pr\left[\begin{array}{l} V(pp, d, \sigma, \\ pk_{cam}, x^*, \pi^*) = 1 \end{array} \;\middle|\; \begin{array}{r} (pp, sk_{cam}, pk_{cam}) \leftarrow G(1^\lambda) \\ r \xleftarrow{\$} \mathbb{F} \\ (d, \sigma) \leftarrow \texttt{C2PA.Sign}(sk_{cam}, I, r) \\ (x^*, \text{state}) \leftarrow P_0^*(pp) \\ \pi^* \leftarrow P_1^*(\text{state}) \end{array}\right] -$$
$$\Pr\left[\begin{array}{l} \texttt{PDF417.VerifyDecode}(\\ \quad I^*, x^* \\ ) = 1 \\ \texttt{C2PA.DigestVer}(\\ \quad d, r \\ ) = I^* \end{array} \;\middle|\; \begin{array}{r} (pp, sk_{cam}, pk_{cam}) \leftarrow G(1^\lambda) \\ r \xleftarrow{\$} \mathbb{F} \\ (d, \sigma) \leftarrow \texttt{C2PA.Sign}(sk_{cam}, I, r) \\ (x^*, \text{state}) \leftarrow P_0^*(pp) \\ I^* \leftarrow \varepsilon^{P_1^*(\text{state})}(pp, x^*) \end{array}\right] \leq \text{negl}(\lambda)$$

Definition B.4 - Zero knowledge. The verifier should not learn anything about the image except that it contains a PDF417 barcode decoding to x. More formally, for all efficient adversaries V^*, there exists an efficient simulator S^{V^*} such that for all efficient distinguisher algorithms D:

$$\Pr\left[\begin{array}{l|r} & (pp, sk_{cam}, pk_{cam}) \leftarrow G(1^\lambda) \\ & r \xleftarrow{\$} \mathbb{F} \\ & (d_0, \sigma_0) \leftarrow \mathsf{C2PA.Sign}(sk_{cam}, I, r) \\ D(pp, pk_{cam}, & (x, aux) \leftarrow \mathsf{PDF417.Decode}(I) \\ d_b, \sigma_b, \pi_b) = b & \pi_0 \leftarrow P(pp, I, x, aux) \\ & (d_1, \sigma_1, \pi_1) \leftarrow S^{V^*}(pp, sk_{cam}, x) \\ & b \xleftarrow{\$} \{0,1\} \end{array}\right] \leq \frac{1}{2} + \mathrm{negl}(\lambda)$$

C Sub-Circuit Details

This appendix provides details on the circuits that we elided for length in the main body.

C.1 Binarize

This sub-circuit converts from the full, greyscale image to a binarized image of just the barcode. The prover provides the binarized image as an input in the augmented witness. The circuit then shows consistency between the grayscale image and the binarized image. This requires two steps for each pixel in the barcode:

1. Assert that the binarized image pixel really is binary: $\texttt{bin\_image}[\texttt{r}][\texttt{c}] \cdot (1 - \texttt{bin\_image}[\texttt{r}][\texttt{c}]) == 0$

2. Assert that the binarized value is consistent with the grayscale image with a lookup table. Essentially, we look up black pixels (binary 1's) in a table from $[0, 127]$, and white pixels in $[128, 255]$, but we can do this with only one lookup table by subtracting 128 from the white pixels.

   (a) Compute the value to lookup: $\texttt{to\_lookup} = \texttt{image}[\texttt{r}][\texttt{c}] - 128 \cdot (1 - \texttt{bin\_image}[\texttt{r}][\texttt{c}])$
   If the binarized image is 1, look up the image value as-is. If it is a zero, look up the pixel value minus 128.

   (b) Look up $\texttt{to\_lookup}$ in a lookup table containing $[0, 127]$.

Since the image coordinates of the barcode are public, the circuit only iterates over the barcode section of the image and ignores the rest.

C.2 Normalization

Normalization accounts for image resolution in block measurements. It converts block measurements in *pixels* to measurements in *modules*, or logical pixels. It also groups blocks into *words*, or groups of eight blocks.

According to the PDF417 spec, there should be exactly 17 modules per word. Using this fact, normalization works as follows (Figure 9):

1. Place 17 evenly-spaced *sample points* in the 8-block pixel array.

2. Count how many sample points land in each block; this number becomes the length of that block in modules.

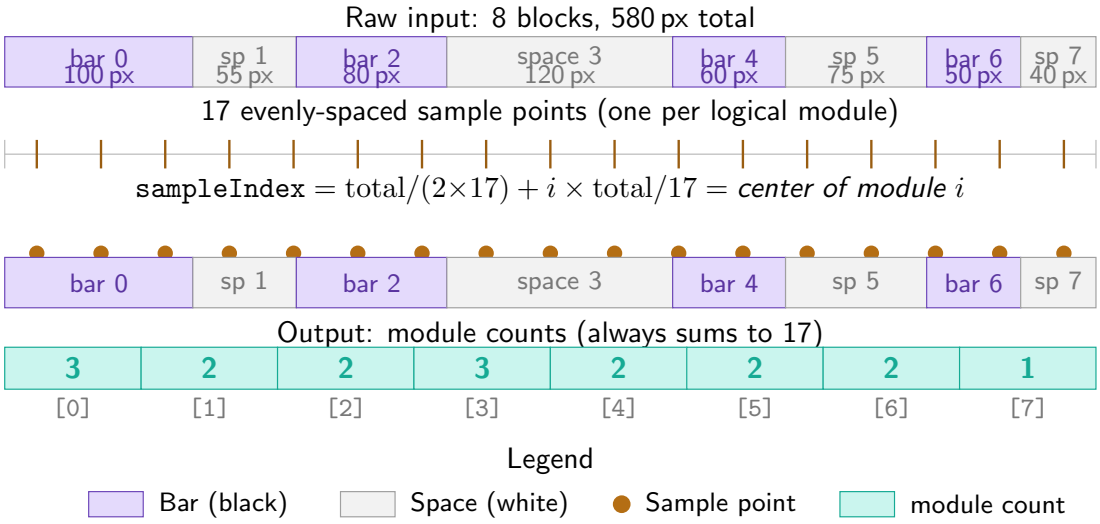


Figure 9: Normalization algorithm

This method ensures that there are always exactly 17 modules in each word.

Instead of running this algorithm directly in circuit, let the prover provide the normalized block measurements as part of the augmented witness, and check that the algorithm has been applied correctly. At each block boundary, verify that the cumulative pixel sum falls between two sample points consistent with the prover-provided block length in modules. This is equivalent to checking that the boundary is in the right place for exactly `normalized_blocks[i]` sample points to have landed in bar i. The circuit computes the position of the last sample point in the block and checks that it is before the pixel boundary. Then, it computes the position of the next sample point and checks this is after the pixel boundary. Inequalities are checked with subtraction and range checks.

The circuit must also handle zero-padding. Since the exact number of blocks is secret, the block measurement circuit pads the array with zeros to get it to a constant length. Normalization checks for zero-length blocks, and if it finds any, it expects all values in the normalized word to be zero.

Once the normalized lengths are checked, words are combined into one field element. Since the word must be exactly 17 modules long and all module lengths must be positive, no single block length can be greater than 17 modules. Thus, we can pack all eight block lengths into one field element base-18.

C.3 Barcode metadata

Use the division algorithm to verify relationships between the row indicators and the barcode metadata according to equations found in the PDF417 spec. Along with the usual equality and range check constraints, each division algorithm must also check that the quotient value provided does not overflow the field modulus when multiplied, as this would allow the prover to cheat.

Let $\texttt{l0}, \texttt{l1}, \texttt{l2}$ be the first three left row indicators from the circuit in Section 5.4. The prover then provides the rest of the values for the system of equations in the augmented witness.

The circuit checks that all of the following equalities and range checks hold:

1. $\texttt{l0} = 30 \cdot \texttt{q0} + \texttt{r0}$ such that $\texttt{q0} \in [0, 928]$ and $\texttt{r0} \in [0, 29]$
2. $\texttt{l1} = 30 \cdot \texttt{q1} + \texttt{r1}$ such that $\texttt{q1} \in [0, 928]$ and $\texttt{r1} \in [0, 29]$

3. $\texttt{l2} = 30 \cdot \texttt{q2} + (\texttt{num\_cols} - 1)$ such that $\texttt{q0} \in [0, 928]$ and $(\texttt{num\_cols} - 1) \in [0, 29]$

4. $\texttt{num\_rows} - 1 = 3 \cdot \texttt{r0} + \texttt{r1} - 3 \cdot \texttt{ec\_level}$ such that $(\texttt{r1} - 3 \cdot \texttt{ec\_level}) \in [0, 2]$

5. $\texttt{ec\_level} \in \texttt{range\_ec\_level}$

6. $\texttt{num\_rows} \in \texttt{range\_num\_rows}$

Lastly, check that `num_ec_cw` is computed correctly from `ec_level` using a lookup table. Assert that $(\texttt{ec\_level} + 1, \texttt{num\_ec\_cw}) \in \texttt{powers\_of\_two}$.

C.4 Error correction

Check that BCH Reed-Solomon error correction has been performed correctly over the codewords extracted from the barcode image. As stated above, let the prover provide the corrected codewords and check correctness by verifying:

1. the error polynomial is sufficiently sparse, and

2. the corrected codeword polynomial evaluates to zero at all relevant evaluation points.

The error polynomial is the codewords polynomial minus the corrected codewords polynomial. Compute the number of non-zero terms by looping over the codewords, computing $\texttt{codewords}[\texttt{i}] - \texttt{corrected\_cw}[\texttt{i}]$, and adding to an accumulator if the result is non-zero. Call this sum e.

If the error correction level is 0, then no errors can be corrected; check that the error polynomial is the zero polynomial. Otherwise, the error polynomial is sparse enough if $\texttt{num\_ec\_cw} \geq 2e + 3$. (This bound is actually slightly too strict - see Section 3.2.3.) To check this, we verify that $\texttt{num\_ec\_cw} - 2e - 3$ can be represented with 9 bits. If not, then the subtraction took us below zero and the field modulus wrapped around, and thus the inequality does not hold.

Next, we need to check that the corrected codeword polynomial evaluates to zero at the specified evaluation points, thus indicating it has been properly corrected. The evaluation points are 3^i $\forall i \in [1, \texttt{num\_ec\_cw}]$, and all arithmetic should be done mod 929 (ie, in the field $GF(929)$). Since `num_ec_cw` is private and variable, we must compute polynomial evaluations for $i \in [1, 512]$ (512 being the maximum number allowed by the PDF417 spec), but then we only use the evaluations that `num_ec_cw` specifies.

For each polynomial evaluation, we need to check that the arithmetic has been done correctly in $GF(929)$. We check the modular arithmetic using the division algorithm. Have the prover provide `result_quotient` and `result` in the augmented witness, where `result_quotient` is the quotient value used in the division algorithm and `result` is the remainder and final answer. For $i \in [1, 512]$, do the following steps:

1. Compute the sum of polynomial terms pre-mod 929. According to the PDF417 spec, the evaluation points are the powers of 3. Since we are working over $GF(929)$, we need to compute $3^i \mod 929$. Use a lookup table of these values. Iterate through the coefficients, and compute each term as $\texttt{corrected\_cw}[\texttt{j}] \cdot \texttt{powers\_of\_three\_mod\_929}[\texttt{i} \cdot \texttt{j} \mod 928]$ (thus computing $3^{i \cdot j} \mod 929$).Computing that index requires no constraints since it is arithmetic over constant values that can be done at compile time. Then, a constant index into an array of constants is also constant. Sum up these terms. We can be sure this term will not overflow the field modulus because each codeword and each power of 3 is in $GF(929)$, and $929^2 \cdot 2700$ is less than the field modulus.

2. Using `result_quotient` and `result`, apply the division algorithm to check this sum mod 929:

$$\texttt{sum} = 929 \cdot \texttt{result\_quotient}[i] + \texttt{result}[i]$$

   In order for this to be sound, we must check that $\texttt{result} \in GF(929)$ and that `result_quotient` is representable in 22 bits to ensure the prover does not overflow the field modulus. The max of `sum` is $929^2 \cdot 2700$, so the max of `result_quotient` is $929 \cdot 2700$, so a valid value is representable in 22 bits. The lookup table required to do this range check is too large, so we resort to bit decomposition instead.

3. Let the prover provide `should_be_zero` for each evaluation in the augmented witness, and check that it is set correctly. If it is false, check that $i >$ `num_ec_cw` using a lookup table. We do not need to check the true case; it is ok if the prover provides true when it should be false, as this does not affect soundness. It is only important that all false's are correct.

4. Check that the polynomial evaluates to zero if this evaluation point is in the relevant range. Check that either $\neg\texttt{should\_be\_zero}[i]$ OR $\texttt{result} == 0$.

C.5 Barcode size

From the row indicators, we know the number of rows and the number of data columns. The product of these two values should be equal to the number of corrected codewords (including error correction codewords.) Furthermore, the first data codeword is the SLD, which tells how many decode-able codewords there are, including itself: the SLD, data codewords, and pads. This sub-circuit checks these relationships.

Since the circuit must be generic to any size of barcode, the `corrected_cw`[`W`] array may have many zeros for higher-order terms that are not used. The first non-zero element (working from high-to-low degree coefficients) is the SLD. To count the number of codewords (including error correction codewords), the circuit counts the number of terms in the array after and including the SLD. The circuit also notes the value of the SLD.

With the SLD and this count in hand, the circuit compares the total count with `num_rows` · `num_cols` passed as input (previously extracted from row indicators). It also computes the number of error correction codewords as $2^{\texttt{ec\_level}+1}$, and asserts that $\texttt{SLD} == \texttt{count} - \texttt{num\_ec\_cw}$.