Efficiently parsing existing eID documents for zero-knowledge proofs

Tom Godden[1], Ruben De Smet[1,2], Kris Steenhaut[1,2] and An Braeken[1]

[1] Vrije Universiteit Brussel, Department of Engineering Technology (INDI), Belgium
[2] Vrije Universiteit Brussel, Department of Electronics and Informatics (ETRO), Belgium

Abstract. Online services increasingly require users to verify their identity or parts of it, often by law. This verification is usually performed by processing data from official identity documents, like national identity cards. However, these documents often contain significantly more information than the verifying party needs to know, including information that should stay private. Disclosing this information is a significant privacy and security risk for the user. Traditional work has designed selective disclosure and zero-knowledge proof protocols for such use cases. However, because these require a complete reimplementation, recall and redistribution of existing identity documents, they have never been adopted on a large scale. More recent work has focused on creating zero-knowledge proofs from existing identity documents like the US passport or specific US driver licenses. In this article, we propose an R1CS protocol to efficiently parse and extract fields from existing European National Identity Cards, with an implementation for the Belgian BeID. The protocol is able to prove correct extraction of a date-of-birth field in 22 seconds on a consumer device, with verification taking 230 milliseconds. With this, we aim to provide EU citizens with a practical solution to the privacy and security risks that arise when one has to prove their authenticity or authority to a third party.

Keywords: zero-knowledge proofs · zk-SNARK · privacy · eID · age verification · self-sovereign identity · anonymous credentials

1 Introduction

Online age or identity verification has become a frequently-discussed topic in recent years. More and more governments [Eur24; Dep24; pax23] are enacting or discussing laws that demand online service providers of e.g. social networks or pornographic websites to verify the age or identity of their users. Various social networks also provide the ability for users to verify themselves voluntarily, so that they can be flagged as an authentic user.

Currently, in order to verify their identity or age, users generally have to send a scan or picture of an official identity document to the service provider. The service provider then verifies this document and subsequently allows or denies access to the user. However, with this kind of verification come severe privacy risks.

First, the user discloses significantly more information than is generally necessary. For example, to prove that Alice is an actual person, she only has to show that she has a legitimate identity card, not the information on the card. Bob might want to prove his the authenticity of his credentials by disclosing just his name, nothing else. Carol might

This study was performed within the framework of Innoviris 2024-RPF Sufficiency and data minimization (SDM) and Cybersecurity Research Program Flanders - second cycle (VOEWICS02).

E-mail: tom.godden@vub.be (Tom Godden), rubedesm@vub.be (Ruben De Smet), Kris.Steenhaut@vub.be (Kris Steenhaut), an.braeken@vub.be (An Braeken)

only want to disclose to a service provider that she is of age to use the service, not her entire identity or even her date of birth. To add to this, certain information printed on the card (e.g. the person's national identity number) should only be known to the issuer and the holder. Second, this makes these users vulnerable to potential identity theft. If these online services were to be compromised, important identity information could easily be accessed by a malicious party. Third, this approach dismantles online anonymity, because the user will always have to link their real identity to their account at least once. Last, on such a picture, a proof of authenticity is often not clearly visible, which means that the picture can be tampered with.

One solution to this could be to use the digital information that is often found on these identity documents. Indeed, this data is used by government services themselves. However, the proof of authenticity on these files applies to the entire file, which means that these files have to be disclosed in their entirety if one wants to verify their authenticity. Consequently, this would lead to the user disclosing even more sensitive data. Furthermore, the digital data on these documents contains significantly more information than the printed-on data: e.g. the Belgian Identity Card (BeID) contains an identity file with identity information about the holder, but also e.g. information about the holder's citizenship status, and it also contains an address file. Whereas this data can be very useful for certain use cases, it can't be used without a full disclosure of the rest of the file.

1.1 Digital wallets

An alternative to the aforementioned disclosure of data is federated identity management with a digital wallet. Recently, the EU has announced that it is developing a "European Wallet" [Eur25]. However, different civil rights associations have expressed their concerns with the proposed wallet, citing potential risks to privacy and personal security [ESH+23], as well as insufficient security on the wallet itself [Eur].

Proprietary solutions to digital wallets exist. For example, the most commonly used digital wallet in Belgium is a proprietary for-profit application called "ItsMe" [itsnd]. Generally, these applications work with users sending their identity information to the wallet provider. When a user wants to use an online service that requires authentication, the service will contact the wallet provider and request authentication. The wallet provider will then tell the user which data they will disclose to the service in order to authenticate the user, which the user then has to confirm or deny.

There are many issues with these kinds of implementations. First, the user needs to put a lot of trust in the wallet provider. They share important sensitive data with the wallet provider and must trust in the correct handling of this data. However, most of these applications are proprietary and for-profit, which means trust is hard to earn. Furthermore, the user does not really have a choice in what wallet provider to use: often the service only implements one type of wallet. This means that, when a user is not satisfied with the service provided by the wallet provider, they cannot simply go to a competitor. Second, the wallet provider simply discloses the requested data to the service provider. By doing this, they disclose significantly more information than the required "this user is allowed to do this". Depending on the wallet provider, the user does get a prompt to confirm the data disclosure. However, they cannot finely tune the data that is provided. Third, as a result of these previous points, we can say that the user has no sovereignty over their data. They do not own the data, they do not control the flow of the data and they don't necessarily know what is happening with the data.

In order to ensure user control and sovereignty over their data, the user needs to be in control of the data flow. This is more or less the case with the physical identity cards: the user holds the only copy of the data, with the original data on the government's servers. Evidently, a user does not have full control over their data, since they are not allowed to edit their official identity information, but for the purpose of data flow, they are in control.

As explained above, the issue with these identity cards is the disclosure of the data on these cards. Therefore, research has been done for privacy-preserving identity documents.

Historically, some different approaches to privacy-preserving eID cards have been proposed: e.g. Vullers and Alpár [VA13] and Mostowski and Vullers [MV12] discuss, respectively Idemix and U-Prove on eID cards. However, these approaches require governments to issue new identity documents, workflows and infrastructure to process these. Given the fact that these technologies have been around for over a decade and no government has announced plans to implement them yet, it seems unlikely that this will happen in the near future. Because of this, recent work has focused on using existing documents, namely the US passport [RWGM23] or US mobile driver licenses [FS24].

1.2 Zero-knowledge proofs from existing credentials

Rosenberg, White, Garman, and Miers [RWGM23] propose zk-creds, a protocol that allows holders of US passports to perform zero-knowledge proofs of knowledge (ZKPs) using their US passports. To use zk-creds, a passport holder first registers their passport on a bulletin board, e.g. a blockchain or a centralized server, in a way that keeps the data secret. They do this by transforming their passport into a merkle tree where each field in the passport is hashed to a leaf node in the merkle tree. They then disclose the root of this merkle tree, which will be stored on the bulletin board. To prove correctness of this transformation without revealing any information, the user creates a ZKP that proves that their passport is equivalent to the merkle tree (without disclosing either), that the merkle tree is constructed correctly and that the root of the merkle tree matches the disclosed root. To prove authenticity, they disclose the signature of the passport and create a ZKP that verifies this signature using the known public key and hidden passport. In order to extract and use a data field of the passport, a user creates a ZKP that proves that they know a merkle tree whose root node matches a root node in the bulletin board, and that this field hashes to a leaf node in this merkle tree.

One obvious issue with this approach is the following: the signature of a passport is unique for each passport. This means that disclosing this signature to prove authenticity directly links the root node on the bulletin board to the passport. In a world where only zk-creds is ever used, this is not a severe issue. However, realistically, passports will be used in a conventional manner, where data is disclosed and the signature is verified in plaintext. Consequently, the holder of the passport can be permanently linked to the root node after just a single conventional use of the passport, effectively deanonymizing the entry on the bulletin board. The reason that zk-creds disclose the signature instead of proving the authenticity of the document in zero-knowledge, is the fact that the signature algorithm used on US passports, ECDSA on secp256r1 with SHA-256 for hashing, is extremely inefficient when transformed to zero-knowledge constraint systems.

Recently, Frigo and Shelat [FS24] have proposed several optimizations to significantly decrease the overhead of ECDSA with SHA-256 for ZKPs, with the specific goal of increasing the usability of existing identity documents for ZKPs. In this article, they achieve sub-second proving and verification times for ZKPs on certain US driver licenses. This makes it realistic to implement privacy preserving applications using these US driver licenses. It is possible to apply these optimizations to both the US passport in the context of zk-creds, or to the European eID cards in the context of this article. This would greatly increase the performance of verifying the authenticity of these documents for ZKPs.

However, another important reason for Frigo and Shelat's very fast proving and verification times is the fact that MDOC, the data format of the US driver licenses they discuss, supports verifiable selective disclosure. In contrast, the US passport uses the machine-readable passport (MRP) format and the European electronic identity cards (eIDs) use the tag-length-value (TLV) format, both of which do not support verifiable selective disclosure. It should be noted that, in principle, the BeID supports selective disclosure;

however, there is no proof of authenticity attached to the disclosed values, so this feature is unsuitable for any real-world applications. This has two detrimental consequences: first, the entire identity document has to be allocated in the constraint system, which gives a slight overhead. Secondly, and most importantly, the identity documents need to be parsed in a constraint system. This is rather simple for the MRP-formatted US passport, since each field in this document has a fixed size. To extract a field, one simply has to pick the correct substring of bytes of which the location and length are known beforehand by both the prover and the verifier. The eID however has fields of variable length, which means the constraint system would need to parse the identity document.

1.3 Zero-knowledge parsing

Most sufficiently powerful parsers perform large numbers of nested branches and loops. This makes it difficult to program an efficient parser in constraint systems like a rank-one constraint system (R1CS) or an arithmetic circuit, which are the constraint systems processed in most zero-knowledge succinct non-interactive arguments of knowledge (zk-SNARKs). In brief, constraint systems do not support flow, because a branching operation does not exist. Instead of branching based on a conditional, both branches are always computed in the following way: $(condition \wedge consequent) \vee (\neg condition \wedge alternative)$. This leads to a second issue, namely that arbitrary stop conditions are not possible, since this requires a branch operation. Instead, loop unwinding needs to be performed, which means that only loops with a fixed number of iterations are possible. Because of these previous two points, branching in loops leads to an exponential explosion of constraints.

Therefore, a naive approach to a TLV parser, where the constraint system parses the witness data, is extremely inefficient. However, because ZKP are proofs of knowledge, it is very often possible to write a constraint system that verifies the proof of knowledge, instead of computing the result. We use this philosophy to create an efficient parser for TLV-formatted data. The prover will generate metadata from the plaintext data, which will then be used to reduce the size of the constraint system, leading to significantly faster proving times. The protocol is described in section 3.

Frigo and Shelat discuss a parser for MDOC-formatted data, which is encoded as Concise Binary Object Representation (CBOR), which can have variable length fields. Therefore, they encounter similar issues and apply similar solutions for their parser. However, since CBOR and TLV are inherently different data formats, we design a completely distinct parser. Not only the field format itself is different, but there are more significant key differences: for example, CBOR features recursion and arrays where TLV does not. On the other hand, TLV can have padding, whereas CBOR cannot.

1.4 Zero-knowledge authentication

A traditional authentication protocol consists of the following parts, all of which need to be verified successfully, after which the owner will be authenticated:

Disclosure of credentials: the holder of an identity document reveals (part of) their data to the service provider.

Processing credentials: the service provider processes the credentials on the identity document and verifies whether these would allow or grant permission to the holder.

Verifying authenticity: the service provider verifies the proof of authenticity, which is generally a signature.

Verifying revocation: the service provider verifies that the document has not been revoked, generally by comparing it to a revocation list.

For a privacy-preserving application we want to hide the document, the extracted values and the signature, and only reveal whether or not a user is allowed to do something. To recreate this workflow in a privacy-preserving way in a zk-SNARK, the R1CS will

consist of four "gadgets", following the same structure as the traditional protocol. A gadget is similar to macros in classical programming languages: it will be expanded to a set of R1CS constraints with a certain purpose. The four distinct gadgets are the following:

Extracting credentials to witness variables: the credentials need to be extracted from the relevant data format without revealing them.

Processing credentials: the prover can then prove that they adhere to a set of rules, based on the extracted credentials.

Verifying authenticity: the prover can generate a proof of correctness about the signature validation of the certificate. This way, she can prove authenticity without disclosing the signature to the verifier.

Verifying revocation: similarly, the prover can prove that their document is not in the revocation list.

This article focuses on the first gadget, discussing the parsing and extraction of values from the BeID. The second gadget can evidently be of any complexity, depending on what criteria are required for authentication. The most simple variant of this gadget simply discloses the value and then proves that the disclosed value is the same as the extracted value, which will allow for selective disclosure on the document. A more complex example is age verification. For specifically the BeID, the prover embeds the current date minus 18 years in `YYYYMMDD` format (e.g. 20070101). Then she converts the extracted date, which is originally in `YYYY.MM.DD` format to the `YYYYMMDD` format and constraints that this extracted value is lower than the embedded value. This also only adds a negligible number of constraints and thus has a negligible effect on performance. For the third gadget, which verifies authenticity of the certificate, we refer to Frigo and Shelat [FS24] for an efficient implementation of ECDSA for use in zk-SNARKs, with sub-second proving time. The last gadget, the revocation verification gadget, is non-trivial and is briefly discussed in section 6.1.

2 Preliminaries

2.1 TLV Format

Our concrete implementation is based on the official documentation of the Belgian Identity Card (BeID) [Fed25]. The Belgian Identity Card (BeID) contains several data files, some of which are encrypted, some of which are plaintext. The most important files for our use case are the identity and address files. These files contain information respectively about the users' identity, including their name, date of birth and picture, and their address. There are two options of querying these files: either on a per-field basis, or by loading the entire file. However, there is only a proof of authenticity for the entire file, meaning that the per-field querying is more or less irrelevant. For this reason, we need to base our protocol on the entire files. We can extract these files from the BeID using official free and open source software [Fed25]. For our protocol, only these files are required, not the identity card itself.

The BeID files contain data in a "simplified TLV" format. A TLV file consists of a series of entries, which each consist of three fields: a tag field indicating the type of the field, a length field referring to the length of the value field, and said value field. The meaning of each tag can be found on the official documentation. The length field consists of any number (including 0) of bytes with value 255, terminated with a byte of any other value than 255. The total length of the value field is equal to the sum of these individual length bytes. The value field is a series of bytes whose length is indicated by the length field. Between tags and at the end of the file, there can be padding in the form of any number of bytes with value 0. To illustrate this, Figure 1 shows the first bytes of an example BeID address file.

01	0c	30	30	30	30	30	31	31	31	32	31	36	33	02	10
53	4c	42	50	03	46	01	06	86	4c	44	14	12	92	12	00
03	0a	32	33	2e	30	39	2e	32	30	32	31	04	0a	32	33
2e	30	39	2e	32	30	33	31	05	07	42	72	75	73	73	65

Figure 1: The first 128 bytes of the BeID test card address file in hexadecimal representation. The alternating background color indicates the individual TLV entries. The first byte `0x01` is the tag of the first TLV entry, which indicates that the entry contains Card Number data. The second byte `0x0c` declares that the following `0x0c` = 12 bytes are the value field. The value field of the Card Number data is formatted as ASCII, so the card number for this card is 000001112163. After these bytes follows the next TLV entry with tag `0x02` (Chip Number).

It is important to note that, although the standard supports it, the actual BeID does not seem to have any fields longer than 254 bytes, nor does it seem to use padding. This will lead to certain optimizations discussed in section 3.

2.2 Sensitive Data and Metadata

One can argue about what is classified as sensitive data and metadata. In our protocol, we classify sensitive (meta)data as everything that gives information about the non-disclosed data. Our goal is to have all sensitive data and metadata as secret witnesses to the proof, so that the verifier does not learn anything about them. We believe this is important because disclosing several small pieces of metadata can lead to different uses of a document being linkable, which is evidently undesirable.

Important sensitive data includes the data file, the extracted TLV triplet and the other TLV triplets. Important sensitive metadata includes the length of the data file, the metadata file M and the index and length of the disclosed triplet in the file.

The index of the disclosed tuple gives information about the length of previous triplets, so we view it as sensitive data. In our protocol, we will therefore consider it secret witness data. However, this also makes our protocol significantly more complex.

We can hide the true length of the extracted triplet by padding it with garbage data and passing its actual unpadded length as witness data to the proof. The only piece of sensitive information we cannot hide is the total length of the data file. The R1CS needs to know how many variables to allocate for the data file. We cannot easily add extra padding to the data file, because then a proof of authenticity would fail.

2.3 Definitions

We give some definitions to be used in the following formulae. We define TLV entries as follows:

TLV Triplet T_i $= t_i l_i v_i$, with t_i, l_i and v_i respectively the byte sequences of the tag field, length field and value field. Both the length and value field can span multiple bytes; the tag field is always one byte.

TLV metadata for T_i: $M_i = (I_i, \Lambda_i, L_i)$, with

- I_i**:** the index of the triplet in the data file D
- Λ_i**:** length of the length field
- L_i**:** value of the length field

Witness Data: the following is witness data, i.e. data that only the prover has knowledge of:

Data File D $= d_0 d_1 \ldots$ is the byte sequence of the full TLV file

$\boldsymbol{I_x}$**:** index of the first byte of T_x in D from which can be computed:
$\boldsymbol{M}$ $= \{M_0, M_1, \ldots, M_{m-1}\}$: metadata file. The prover generates this file beforehand (on a Turing machine, not a R1CS).
$\boldsymbol{T_x}$ $= t_x l_x v_x$: the extracted TLV entry

Public Data: the following data is public data, i.e. data of which both the prover and the verifier have knowledge:
$\boldsymbol{|D|}$**:** the total length of data file D
$\boldsymbol{m}$**:** number of TLV entries in data file D
$\boldsymbol{\lambda_x}$**:** The padded length of T_x
Proof settings :

- $\boldsymbol{\Lambda}$: optional fixed size for length of the length field
- whether or not padding is used

3 Circuit Design

The constraint system is designed with the following high-level constraints:

1. The extracted TLV triplet is a substring on the file.
2. The metadata is correct. For this, the prover proves that:
   (a) Metadata starts at 0
   (b) Metadata length fields are parsed and calculated correctly
   (c) TLVs don't overlap
   (d) Padding between TLVs is correct
3. The extracted triplet is consistent with the metadata.

The following sections describe these constraints in detail. Pseudocode for the protocol can be found in algorithm 1. For consistency and clarity, we use i to iterate over bytes of the data file D and j to iterate over TLV metadata triplets from M.

3.1 Substring (constraint 1)

We constrain that the extracted triplet T_x actually occurs in the file D. In other words, T_x must be a substring of D.

String a is a substring of string D if there exist a prefix string p and suffix string s so that $D = pas$. We create a new string $\Delta = \delta_0, \delta_1, \ldots$, with $|\Delta| = |D|$, as follows:

$$\delta_i = \begin{cases} 2^{8(i-I_x)} d_i & \text{if } (i \geq I_x) \wedge (i < I_x + \Lambda_x + L_x + 1), \\ 0 & \text{otherwise} \end{cases}$$

This achieves two goals:

1. it makes every byte in prefix p and suffix s 0
2. each byte value of substring a is multiplied with a unique power of 2^8

This means that $\sum_{\delta \in \Delta} \delta$ is equal to the concatenation of all bytes at the selected indices.

We perform a similar computation on the padded T_x to compute $\Theta = \theta_0, \theta_1, \ldots$ with $|\Theta| = \lambda_x$:

$$\theta_i = \begin{cases} 2^{8i} \tau_i & \text{if } (i < \Lambda_x + L_x + 1), \\ 0 & \text{otherwise} \end{cases}$$

$\sum_{\theta \in \Theta} \theta$ is thus equal to the concatenation of all bytes of T_x.

From this follows that T_x is indeed a substring of D at location I_x if:

$$\sum_{\delta \in \Delta} \delta = \sum_{\theta \in \Theta} \theta$$

Algorithm 1 Pseudocode for the eID parser. Definitions of the input values can be found in section 2.3.

▷ *Constrain substring (constraint 1)* ◁
for all $i \in [0, \ldots, |D|)$ **do**
  $\delta_i \leftarrow 2^{8(i-I_x)} d_i \wedge (i \geq I_x) \wedge (i < I_x + \Lambda_x + L_x + 1)$
  $\theta_i \leftarrow 2^{8i} \tau_i \wedge (i < \Lambda_x + L_x + 1)$
  Constrain: $\sum_{\delta \in \Delta} \delta = \sum_{\theta \in \Theta} \theta$
▷ *Metadata starts at 0 (constraint 2a)* ◁
Constrain: $I_0 = 0$
▷ *Length fields (constraint 2b)* ◁
if length fields have variable length **then**
  ▷ *Length field formatting* ◁
  for all $i \in [0, \ldots, |D|)$ **do**
    is_l_pref $\leftarrow \exists j \in [0, \ldots, m) : i \in [I_j, ...I_j + \Lambda_i)$
    is_l_suf $\leftarrow \exists j \in [0, \ldots, m) : i = I_j + \Lambda_i$
    Constrain: (is_l_pref $\wedge$ $(d_i \neq 255)$) $\vee$ (is_l_suf $\wedge$ $(d_i = 255)$) $\vee \neg$(is_l_pref $\vee$ is_l_suf)
  ▷ *Length field values* ◁
  for all $j \in [0, \ldots, m)$ **do**
    $s \leftarrow 0$
    for all $i \in [0, \ldots, |D|)$ **do**
      $s \leftarrow s + (x_i \wedge i \in [I_j, ...I_j + \Lambda_i))$
    Constrain: $s = L_j$
else
  for all $j \in [0, \ldots, m)$: **do**
    ▷ *Length field lengths* ◁
    Constrain: $\Lambda_j = \Lambda$
    ▷ *Length field values* ◁
    $s \leftarrow 0$
    for all $i \in [0, \ldots, |D|)$ **do**
      $s \leftarrow s + (x_i \wedge i \in [I_j, ...I_j + \Lambda))$
    Constrain: $s = L_j$
▷ *TLV's don't overlap (constraint 2c)* ◁
for all $[0, \ldots, m-1)$ **do**
  $I_j + \Lambda_j + L_j < I_{j+1}$
▷ *Padding is correct (constraint 2d)* ◁
if Padding is used **then**
  for all $i \in [0, ..., |D|)$ **do**
    part_of_tlv $\leftarrow 0$
    for all $j \in [0, \ldots, m)$ **do**
      part_of_tlv $\leftarrow$ part_of_tlv $\vee$ $((i \geq I_j) \wedge (i < I_j + 1 + \Lambda_j + L_j))$
    Constrain: part_of_tlv $\vee$ $(d_i = 0)$
else
  for all j *in*$[0, \ldots, m)$ **do**
    Constrain: $I_j + 1 + \Lambda_j + L_j = I_{j+i}$
  Constrain: $I_{m-1} + \Lambda_{m-1} + L_{m-1} = |D|$
▷ T_x *is consistent with the metadata (constraint 3)* $f \leftarrow 0$ ◁
for all j *in*$[0, \ldots, m)$ **do**
  $f \leftarrow f \vee (I_x = I_j)$
Constrain: f

A note about branching Note that, as mentioned before, we want to avoid as many branches in loops as possible. The computation of Δ might seem likely exactly that, but, since our comparisons return either 0 or 1, we can calculate δ_i as follows:

$$\delta_i = 2^{8(i-I_x)} d_i \times (i \geq I_x) \times (i < \Lambda_x + L_x + 1)$$

which does not contain any branches. The same holds for the computation of Θ.

3.2 Metadata starts at 0 (constraint 2a)

We constrain that the first index of the metadata starts at index 0, so that no bytes at the front of the file can be skipped:

$$I_0 = 0$$

3.3 Length fields (constraint 2b)

As explained in section 2.1, TLV supports fields of arbitrary lengths, but it does not seem like the identity and address files of the BeID have any fields that would realistically make the length of the length field Λ_i greater than one byte. Therefore in paragraph 3.3.1, we first discuss the general case where the length field can have variable lengths. Then, in paragraph 3.3.2 we discuss the case where length fields only have a fixed length, which will generate significantly fewer constraints for our constraint system. If the prover does not want to disclose whether or not their BeID has length fields of different lengths, they can choose the general case, even if all length fields have an equal length.

3.3.1 Λ_i is variable

For the length L_i in each $M_i \in M$ we constrain that the length must be formatted correctly and that the value of the length field must match the witness value in M_i.

Formatting: A length field is formatted correctly if every symbol is 255, except the last one which mustn't be 255.

We constrain:

$$\forall j \in [0, \ldots, m) : \begin{cases} \forall k \in [I_j + 1, ..., I_j + \Lambda_j + 1) : d_k = 255, \\ d_{I_j + \Lambda_j + 1} \neq 255 \end{cases}$$

Value: Since we have constrained that the length is formatted correctly, the value of the length field is simply the sum of all bytes.

We constrain:

$$\forall j \in [0, \ldots, m) : L_j = \sum_{i=I_j+1}^{I_j+\Lambda_j+1} d_i$$

3.3.2 Λ_i is fixed:

For each TLV triplet we constrain that the length of the length field from the metadata witness must match the publicly declared value Λ.

$$\forall j \in [0, \ldots, m) : \Lambda_j = \Lambda$$

We also constrain that for each TLV triplet, the value of the length field must match the value in the corresponding metadata:

$$\forall j \in [0, \ldots, m) : L_j = \sum_{i=I_j+1}^{I_j+\Lambda+1} d_i$$

3.4 TLV's don't overlap (constraint 2c)

To constrain that there must not be any overlapping triplets according to the supplied metadata M, we constrain that the index of the end of the i-th TLV triplet must be strictly smaller than the index of the start of the $i+1$-th index:

$$\forall j \in [0, \ldots, m-1) : I_j + \Lambda_j + L_j < I_{j+1}$$

3.5 Padding is correct (constraint 2d)

As explained in section 2.1, the BeID supports padding between TLVs, but it does not seem like this is used. If no padding is used, we can optimize our constraint system to require significantly less constraints. Therefore in paragraph 3.5.1, we first discuss the general case where padding is possible. Then, in paragraph 3.5.2 we discuss the case where no padding is used. If the prover does not want to disclose whether or not their BeID is using padding, they can choose the general case, even if no padding is used.

3.5.1 Padding 0

Padding is each symbol which is not between the start and end of a TLV tag. All padding must to be 0, so that no data can be "hidden" outside of any TLV triplet.

We constrain that each element d not in the bounds of any TLV triplet must be 0:

$$\begin{gathered}\forall i \in [0, \ldots, |D|) : \\ \text{if } \nexists j \; in [0, \ldots, m) : (i \geq I_j) \wedge (i < I_j + 1 + \Lambda_j + L_j) \\ \text{then } d_i = 0\end{gathered}$$

3.5.2 No padding

To enforce that no padding is used we constrain that the end index of every TLV triplet must be one less than the start index of the next one:

$$\forall j \in [0, \ldots, m) : I_j + 1 + \Lambda_j + L_j = I_{j+i}$$

and that the last TLV triplet ends at the end of the file:

$$I_{m-1} + \Lambda_{m-1} + L_{m-1} = |D|$$

3.6 T_x is consistent with the metadata (constraint 3)

To ensure that the substring starts in a location that is consistent with the metadata, we constrain that the start index of the extracted value must be one of the start indices of M:

$$\exists j : I_x = I_j$$

4 Performance

4.1 Benchmarking environment

We have benchmarked the number of constraints in the eID parsing and extraction gadget, proving time and verification time across three variables, namely:

- whether the file contains padding or not,
- whether tags have a fixed or variable length
- and whether we perform the cryptographic computations on a single core or multiple cores simultaneously.

Table 1: Number of constraints in the R1CSs. The differences caused by the optimization are highlighted in bold.

L-field length Padding?	Variable ✓	Variable	Fixed (1) ✓	Fixed (1)
Allocate D	1,936	1,936	1,936	1,936
Metadata M	864	864	864	864
Allocate T_x	128	128	128	128
Allocate I_x	16	16	16	16
Allocate L_x	16	16	16	16
Substring (1)	22,585	22,585	22,585	22,585
$I_0 = 0$ (2a)	16	16	16	16
L-field (2b)	**305,352**	**305,352**	**119,979**	**119,979**
No overlap (2c)	665	665	665	665
Padding (2d)	**171,578**	**630**	**171,578**	**630**
Start index (3)	576	576	576	576
Total	**503,732**	**332,784**	**318,359**	**147,027**

The number-of-constraints, proving and verification time measurements are related to each other. A more complex proof has more constraints, which leads to longer proving and verification times. The time measurements are very much dependent on the device the benchmarks are executed on and the proof system used. The time measurements are stochastic, whereas the number-of-constraints measurements are not. The number of constraints is only dependent on the R1CS which models the specific proof. The time measurements are intended to show viability and discuss potential applications in section 5. The number-of-constraints measurement is a means to perform an objective, proof-system independent comparison between our different tests. Furthermore, if one knows the performance of a specific proof system, one is able to predict the expected time the proving and verification would take for a specific number of constraints.

We have implemented our design in the Arkworks [ark22] Rust framework. We used the Spartan proof system [Set20], implemented in the ark-spartan [arkb] Rust crate. Our cryptographic operations were performed on the BLS12_377 curve [BCG+20], implemented in the `ark_bls12_377` [arka] Rust crate. The benchmarks were performed using the criterion Rust benchmarking framework [Hei]. Criterion runs each benchmark at least 20 times. We have benchmarked both single and multicore performance, using Rust's Rayon framework. All benchmarks were performed on a laptop with an Intel i7-1165G7 64-bit processor and 16GB of DDR4 RAM. The code can be found at our GitLab repository [Tom25].

As data for the benchmark, the official BeID test card [Zetnd] was used. In the benchmark, we extract the date-of-birth field from the identity file on the test card, but we could have chosen any field. The identity file has a length of 242 bytes, the extracted date-of-birth TLV has a length of 10 bytes (1 tag + 1 length + 8 value), which is padded to 16 bytes to hide the actual length. The file contains a total of 18 TLV tuples. Although the date-of-birth field has a fixed size of 10 bytes, the performance is independent of whether the field has a fixed or variable size, since we treat each field the same way.

4.2 Benchmark Results

Table 1 shows a breakdown of the constraints generated in each part of the R1CS. As is evident from the results in Table 1, we can significantly reduce the number of constraints with our two optimizations.

Table 2 shows the average time needed to generate the proof and verify it with our setup discussed in section 4.1. It is evident that our optimizations discussed in section 3 have a significant impact on the performance, especially when both are applied. Multicore

Table 2: Proving and verification times. The optimized version is highlighted in bold.

Multicore?	L-field length	Padding?	Proving Time (s)	Proving Std (s)	Verification Time (s)	Verification Std (s)
	Variable	✓	148.86	1.7445	0.51348	0.00038
	Variable		89.607	0.08568	0.44334	0.00043
	Fixed (1)	✓	84.893	0.10695	0.41919	0.00065
	Fixed (1)		**42.843**	0.53883	**0.31675**	0.00076
✓	Variable	✓	73.499	0.01150	0.34438	0.00730
✓	Variable		45.503	0.08993	0.29539	0.01119
✓	Fixed (1)	✓	42.287	0.11486	0.28212	0.00795
✓	Fixed (1)		**21.769**	0.06185	**0.22959**	0.00407

proving times are reduced by 38% with just the padding optimization, by 42% with just the length-field optimizations and by 70% with both optimizations applied.

An important thing to note is that our verification times are very low compared to our proving times. Efficient verification times is one of the main features of the Spartan proof system [Set20].

Also noteworthy is the speedup gained by using multiple cores for proving. Multicore benchmarks were between 40% and 100% faster. The gain is most significant for proving benchmarks, since the overhead of parallelism is less impactful there.

4.3 Comparison with Frigo and Shelat [FS24]

As described in section 1.3, Frigo and Shelat also create a zero-knowledge parser. A strict comparison between their protocol and ours is hard because:

- their protocol parses the CBOR format instead of TLV format,
- their constraint system is an arithmetic circuit, whereas ours is a R1CS
- and they do not present timing benchmarks for the parsed data, only for the selective disclosed data.

However, we believe it is still valuable to compare the number of constraints, to show that we have similar results when parsing variable-length data formats.

The official test eID has a length of 242 bytes, so we will compare it to the CBOR message closest to that length that Frigo and Shelat test, which is 254 bytes. For these cases, our optimized TLV parser has 147,027 constraints, whereas their CBOR parser has 115,602 “quads” (quadratic gates). If we assume that the number of constraints in an R1CS is similar to the number of quadratic gates in an arithmetic circuit, because they both represent multiplications, we can see that our TLV parser has a number of constraints that is comparable to their CBOR parser.

5 Evaluation

The evaluation of our protocol is evidently dependent on the use case. For online user verification, 22 seconds of proving time (disregarding sub-second proving times for other gadgets discussed in section 1.4) is, in our opinion, acceptable for many use cases. This is especially the case if this only has to be performed once, e.g. for a one-time user verification.

In another simple use case, Peggy the prover is going to a bar and ordering an alcoholic beverage, for which she needs to prove she is of age to do so. In this case, Peggy is

probably not carrying her laptop around but is likely performing the computations on her less powerful phone. This will significantly increase the proving times, which is evidently undesirable when e.g. there are people in line behind Peggy. Therefore, performance can be an issue depending on the use case. We propose two ways to potentially improve performance for real-world applications.

First, it is possible for Peggy to send her data to a powerful server, who then computes and returns the proof. Evidently, privacy and security measures have to be taken so that the cloud service provider does not get to know any of Peggy's private data. Technologies with which to accomplish this include secure multi-party computation and trusted execution environments.

Second, if Peggy knows that she is going to need to prove her age, it is possible to generate the proof beforehand. However, this removes all flexibility from the proof: there are different age requirements for different beverages, so Peggy would need to know what she wants to drink well beforehand. Furthermore, this approach can potentially lead to replay attacks: someone could copy her proof and use it for themselves.

A solution to this might be recursive proofs [BDFG20]: Peggy can generate a proof beforehand and then embed the proof of this proof in a new ZKP. She can then reuse the same proof every time she needs to prove her age.

Another potential preprocessing solution is to transform the BeID data to a format that is better suited for a zk-SNARK. This is the approach taken by zk-creds [RWGM23]: they transform the US Passport to a merkle tree, and store the root of this merkle tree on a bulletin board.

6 Future Work

6.1 Certificate Revocation

An important issue with certificates is revocation: when a citizen loses their identity document, loses their citizenship or dies, the identity card should no longer be valid. Revoked certificates IDs are published in a certificate revocation list. Therefore, in order to prove that they have a valid certificate, a user should prove that the certificate on their eID is not in the certificate revocation list. This is, in essence a set non-membership problem, which is notoriously hard to solve efficiently in a R1CS.

Rosenberg, White, Garman, and Miers [RWGM23] work around this issue by publishing the hash of a legitimate certificate on a bulletin board and removing the hash if the certificate has been revoked. However, an authentication application does not necessarily have a bulletin board. Furthermore, publishing this hash can lead to the linkability and deanonymisation issues mentioned above in section 1.2.

Frigo and Shelat [FS24] on the other hand describe certificate revocation as future work.

Lipmaa and Parisella [LP23] propose an efficient ZKP for set non-membership, which could be used to efficiently solve this issue. However, their approach is at the level of the proof system, instead of the constraint system. Therefore, research has to be done to see how well this approach would work for our use case.

7 Conclusion

We have discussed a R1CS protocol that can efficiently parse European eID cards, with an implementation that focuses on the Belgian Identity Card (BeID). This protocol enables us to perform zero-knowledge proofs of knowledge (ZKPs) and selective disclosure on existing identity cards. With our protocol, we can use the official Belgian Identity Card (BeID) with enhanced privacy and without the involvement of the issuer or another third

party actor. This is a significant improvement compared to proposed or in-use identity documents or services, both for privacy and security. We have evaluated the performance of our protocol and have discussed methods to improve or work around performance issues, if necessary. With this protocol, we hope to increase base-level privacy for users of these identity cards, especially since users have sometimes very little choice but to use these cards.

References

[arka] arkworks contributors. Ark_bls12_377, version 0.4.0, arkworks. URL: https://docs.rs/ark-bls12-377/latest/ark_bls12_377/.

[arkb] arkworks contributors. Ark_spartan, arkworks. URL: https://github.com/arkworks-rs/spartan.

[ark22] arkworks contributors. Arkworks zkSNARK ecosystem, 2022. URL: https://arkworks.rs.

[BCG+20] Sean Bowe, Alessandro Chiesa, Matthew Green, Ian Miers, Pratyush Mishra, and Howard Wu. ZEXE: Enabling Decentralized Private Computation. In *2020 IEEE Symposium on Security and Privacy (SP)*. 2020 IEEE Symposium on Security and Privacy (SP), pages 947–964, May 2020. DOI: 10.1109/SP40000.2020.00050.

[BDFG20] Dan Boneh, Justin Drake, Ben Fisch, and Ariel Gabizon. Halo Infinite: Recursive zk-SNARKs from any Additive Polynomial Commitment Scheme. 2020. URL: https://eprint.iacr.org/2020/1536. Pre-published.

[Dep24] Transport Department of Infrastructure. Social media minimum age legislation passed. December 4, 2024. URL: https://www.infrastructure.gov.au/department/media/news/social-media-minimum-age-legislation-passed (visited on 04/11/2025).

[ESH+23] Electronic Frontier Foundation (Global) et al., Letter, June 20, 2023. URL: https://epicenter.works/fileadmin/import/cso-eidas-open_letter_2023.pdf (visited on 04/11/2025).

[Eur] Eurosmart. Low Security in the European Digital Identity Wallet: An Unacceptable Risk for Citizens and Businesses, Eurosmart. URL: https://www.eurosmart.com/wp-content/uploads/2024/09/2024_09_10_Low-Security-in-the-EUDI-WalletV1.2_FINAL.pdf (visited on 04/11/2025).

[Eur24] European Commission. Digital Services Act: Task Force on Age Verification | Shaping Europe's digital future. January 30, 2024. URL: https://digital-strategy.ec.europa.eu/en/news/digital-services-act-task-force-age-verification-0 (visited on 03/06/2024).

[Eur25] European Commission. EU Digital Identity Wallet Home - EU Digital Identity Wallet -. 2025. URL: https://ec.europa.eu/digital-building-blocks/sites/display/EUDIGITALIDENTITYWALLET/EU+Digital+Identity+Wallet+Home (visited on 04/11/2025).

[Fed25] Fedict. Eid-mw, FPS BOSA DG Digital Transformation, May 16, 2025. URL: https://github.com/Fedict/eid-mw (visited on 05/16/2025).

[FS24] Matteo Frigo and Abhi Shelat. Anonymous credentials from ECDSA. 2024. URL: https://eprint.iacr.org/2024/2010 (visited on 04/11/2025). Pre-published.

[Hei] Brook Heisler. Criterion.rs, version 0.4. URL: https://github.com/bheisler/criterion.rs.

[itsnd] itsme®. itsme®, de identiteitsapp. itsme®. nd. URL: https://www.itsme-id.com/nl-BE (visited on 04/11/2025).

[LP23] Helger Lipmaa and Roberto Parisella. Set (Non-)Membership NIZKs from Determinantal Accumulators. In Abdelrahaman Aly and Mehdi Tibouchi, editors, *Progress in Cryptology – LATINCRYPT 2023*, pages 352–374, Cham. Springer Nature Switzerland, 2023. ISBN: 978-3-031-44469-2. DOI: 10.1007/978-3-031-44469-2_18.

[MV12] Wojciech Mostowski and Pim Vullers. Efficient U-Prove Implementation for Anonymous Credentials on Smart Cards. In LNICST, volume 96, pages 243–260, January 1, 2012. ISBN: 978-3-642-31908-2. DOI: 10.1007/978-3-642-31909-9_14.

[pax23] Texas House Bill 1181, December 6, 2023. URL: https://capitol.texas.gov/tlodocs/88R/billtext/html/HB01181H.htm (visited on 04/11/2025).

[RWGM23] Michael Rosenberg, Jacob White, Christina Garman, and Ian Miers. Zk-creds: Flexible Anonymous Credentials from zkSNARKs and Existing Identity Infrastructure. In *2023 IEEE Symposium on Security and Privacy (SP)*. 2023 IEEE Symposium on Security and Privacy (SP), pages 790–808, May 2023. DOI: 10.1109/SP46215.2023.10179430.

[Set20] Srinath Setty. Spartan: Efficient and General-Purpose zkSNARKs Without Trusted Setup. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology – CRYPTO 2020*. Volume 12172, pages 704–737. Springer International Publishing, Cham, 2020. ISBN: 978-3-030-56876-4 978-3-030-56877-1. DOI: 10.1007/978-3-030-56877-1_25.

[Tom25] Tom Godden. BeidExtractionGadget, July 1, 2025. URL: https://gitlab.com/etrovub/smartnets/glycos/tlv-selective-disclosure-test (visited on 07/09/2025).

[VA13] Pim Vullers and Gergely Alpár. Efficient Selective Disclosure on Smart Cards Using Idemix. In Simone Fischer-Hübner, Elisabeth de Leeuw, and Chris Mitchell, editors, *Policies and Research in Identity Management*, IFIP Advances in Information and Communication Technology, pages 53–67, Berlin, Heidelberg. Springer, 2013. ISBN: 978-3-642-37282-7. DOI: 10.1007/978-3-642-37282-7_5.

[Zetnd] Zetes. EAZYSET - YOUR SET OF EID TESTING CARDS. Zetes. nd. URL: https://www.eazysign.be/eazyset-your-set-of-eid-testing-cards (visited on 05/16/2025).