UNIVERSITY OF TARTU
Faculty of Science and Technology
Institute of Computer Science
Computer Science Curriculum

Uuna Saarela

Practical Zero-Knowledge within the European Digital Identity Framework: Implementing Privacy-Preserving Identity Checks

Master's Thesis (30 ECTS)

Supervisor(s): Peeter Laud, PhD
Chris Brzuska, PhD

Tartu 2024

Practical Zero-Knowledge within the European Digital Identity Framework: Implementing Privacy-Preserving Identity Checks

Abstract:

The eIDAS regulation defines standards within the European Union (EU) for electronic identification, authentication, and trust services. Its successor eIDAS2 expands the standard and proposes establishing an ecosystem of interoperable national digital identity services for EU citizens. The new regulation will mandate member states to make these services available, and work is ongoing to define the technologies and protocols that will constitute the European Digital Identity (EUDI) framework. The framework is expected to facilitate many use cases that require a person's identity to be verified with a high degree of assurance. These include opening a bank account and authenticating to online services in different EU countries. It will also be possible to hold and present valid digital documents, such as driving licenses and other permits, medical prescriptions, and professional and educational certificates. Widespread adoption of the EUDI framework would significantly increase the volume of digital identification events, which include sensitive personal data and metadata about when and where credentials are used.

This poses a threat to the privacy of EU citizens who wish to access the many benefits of digital identity services. The eIDAS2 regulation includes a provision for the use of privacy-preserving technologies, which enable performing business logic on data that remains hidden. Zero-knowledge proof systems are one such technology. A zero-knowledge toolchain can be used to obtain a cryptographic proof that some statement is true while revealing nothing else about the statement itself. ZK-SecreC is a language developed by Estonian research and development company Cybernetica for writing zero-knowledge programs, which are then compiled into equivalent circuit descriptions. This thesis explores using ZK-SecreC to program the real-world identity checks performed by Finnish company Talenom in conjunction with their user onboarding process. The input data for the program is assumed to be a valid EUDI credential under the current reference architecture, and as such is formatted according to the ISO/IEC 18013-5 standard for mobile driving licenses. The technical contribution of the thesis is a ZK-SecreC program and two variants thereof that perform the desired identity checks. The runtime of the resulting circuits are benchmarked with two different proving backends. The thesis also describes the steps required to integrate zero-knowledge proofs into the EUDI architecture.

Keywords:
zero-knowledge, digital identity, mobile driving license, eIDAS2

CERCS: P170 Computer science, numerical analysis, systems, control

Praktiline nullteadmustõestus Euroopa digitaalse identiteedi raamistikus: privaatsust säilitavate identiteedikontrollide rakendamine

Lühikokkuvõte:

eIDAS määrus kirjeldab Euroopa Liidus kasutatavaid standardeid elektroonilise identifitseerimise, autentimis- ja usaldusteenuste jaoks. Selle edasiarendus eIDAS2 laiendab neid standardeid ja pakub välja ELi kodanike jaoks koostalitlusvõimelise riiklike digitaalsete identiteediteenuste ökosüsteemi loomise. Uus määrus kohustab liikmesriike tegema need teenused kättesaadavaks ning praegu töötatakse selle nimel, et kokku leppida tehnoloogiad ja protokollid, mis moodustavad Euroopa digitaalse identiteedi (EUDI) raamistiku. Raamistik peaks toetama paljusid kasutusjuhtumeid, mis nõuavad isiku identiteedi kõrge kindlusega kontrollimist, nagu näiteks erinevates ELi riikides pangakonto avamine või võrguteenustele autentimine. Samuti võimaldab EUDI kehtivate digidokumentide haldamist ja esitamist, nagu juhiload ja muud load, ravimiretseptid ning kutse- ja haridustunnistused. EUDI raamistiku laialdane kasutuselevõtt kasvataks märkimisväärselt selliste digitaalsete tuvastuste arvu, mille käigus töödeldakse tundlikke isikuandmeid ning metaandmeid selle kohta, millal ja kus volitusi kasutatakse.

See kujutab endast ohtu digitaalse identiteedi teenustest kasu saada soovivate EL-i kodanike isikuandmete kaitsele. eIDAS2 määrus sisaldab sätet selliste privaatsust säilitavate tehnoloogiate kasutamise kohta, mis võimaldavad töödelda andmeid ilma neid töötlejale avalikustamata. Üheks selliseks tehnoloogiaks on nullteadmustõestused, mille abil on võimalik saada krüptograafiline tõend mingi väite tõesuse kohta, avaldamata mingit muud infot selle väite kohta. ZK-SecreC on Eesti teadus- ja arendusettevõtte Cybernetica välja töötatud keel selliste nullteadmuses tõestatavate programmide kirjutamiseks; need programmid kompileeritakse seejärel samatähenduslikeks loogikalülitusteks. Käesolevas magistritöös uuritakse ZK-SecreC kasutamist Soome ettevõttes Talenom uue kasutaja registreerimisprotsessi käigus tehtava identsuskontrolli realiseerimisel. Programmi sisendandmetena eeldatakse kehtiva etalonarhitektuuri alusel kehtivat EUDI tõendit, mis on sellisena vormindatud vastama digitaalse juhiloa standardile ISO/IEC 18013-5. Lõputöö tehniline osa koosneb ZK-SecreC programmist ja selle kahest variandist, mis viivad läbi vajaliku identiteedikontrolli. Saadud loogikalülituste käitusaega mõõdetakse kahe erineva nullteadmustõestusprotokolle realiseeriva tagasüsteemi abil. Samuti kirjeldatakse magistritöös vajalikke samme, et integreerida nullteadmuse tõestamist EUDI arhitektuuriga.

Võtmesõnad:

nullteadmus, digitaalne identiteet, digitaalne juhiluba, eIDAS2

CERCS: P170 Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine

Contents

1 Introduction

Privacy-preserving technologies refer to technologies that enable the collection, storage, and analysis of data without compromising the privacy of that data. With applications in targeted advertising and influence, predictive analytics, and training neural networks, data holds enormous economic potential, and has been called the oil of the 21st century. [SUP14] However, the mining of this data represents a significant threat to the privacy of involved parties. Privacy-preserving technologies include federated learning, homomorphic encryption, secure multi-party computation, trusted execution environments in hardware, and zero-knowledge proof systems. [CGL20] These technologies make it possible to perform business logic on data that remains hidden.

Legal identity documents, such as passports or identity cards, are among the most valuable and the most sensitive data a person can hold. Traditional paper identity documents have weaknesses: they may be slow or expensive to obtain and can only be authenticated in-person. More personal data than is necessary can be revealed inadvertently when using paper identity documents. For example, to access age-restricted products or services, a person often reveals their birth date, name, and personal identity code to the clerk when proving their age with a driving license. Paper signatures are also easy to forge relative to their electronic counterparts. It is increasingly necessary to supplement or replace paper identity documents with equivalent digital identities.

A digital identity identifies an individual or organisation in the digital world and often serves as a credential to access online services. Many commercial service providers, such as Google, Apple, and Facebook, offer digital identity solutions to manage credentials to many services with one digital identity. This eases user experience and mitigates the risk of using insecure duplicate passwords, but exposes more metadata about the user's online behaviour to these service providers. Services requiring "strong authentication", or authentication of legal identity, are increasingly being brought online. These services may include banking, taxation, voting, and creating electronic signatures for binding contracts. The providers of strong authentication services are varied, and may include banks or teleoperators in addition to governments, as in the Nordics. [Sig21]

The provision of digital identity documents "by social media providers and financial institutions, raise privacy and data protection concerns." [EU21] Governments are an intuitive alternative, and many governments provision digital identity documents to residents or citizens in some form, such as electronic identity cards or mobile identities. [dig24] These can be used to authenticate oneself online and in-person, as well as create digital signatures. Regulations can be used to define what algorithms and hardware constitute a legal electronic identity document. One such regulation is the eIDAS, which stands for Electronic Identification, Authentication and Trust Services. The regulation defines standards for electronic identification and trust services within the EU. [EU14] Its successor, eIDAS2 [EU21], proposes the creation of a European digital identity infrastructure and wallet (EDIW) for EU citizens.

eIDAS2 [EU21] will "mandate cross-border interoperability in the EU and facilitate use cases such as opening bank accounts, making payments and holding digital documents, such as a mobile Driving Licence (mDL), a medical prescription, a professional certificate or a travel ticket." [EU] This national identity can be read by public bodies, companies, and institutions to verify the identity of EU citizens. eIDAS2 allows for the use of privacy-preserving technologies, such as zero-knowledge attestations, in national identity solutions. [EU21] These technologies would allow citizens to benefit from the EDIW without losing control over their personal data.

Zero-knowledge proof systems were introduced in 1989 by Goldwasser, Micali and Rackoff [GMR89]. They allow for proving the veracity of a statement while revealing nothing about the statement itself. They can also produce a short proof of a long statement, and so have applications in solving scalability problems [BFL90]. Several practical zero-knowledge systems have been created in recent years, as shown in Table 2. A major application for zero-knowledge proofs has been found in blockchain networks, which "consist of mutually distrusting parties that wish to transact, or generally update collective state according to state evolution rules, using secret information." [Asz] The potential of using zero-knowledge toolchains in other real-world systems, such as digital identity management, is in the process of being realized.

1.1 Thesis Structure

This thesis details the process of using ZK-SecreC, a zero-knowledge DSL, to construct practical proofs of identity according to a mobile driving license specification. It also explores the legal, political, and technical considerations of adopting such technologies in a large-scale identity architecture like the European Union Digital Identity (EUDI) framework. Chapter 1 provides background information and preliminaries. Chapter 2 explores related work on other languages for verifiable computation and zero-knowledge programming. Chapter 3 describes ZK-SecreC and presents some of its unique language features. Chapter 4 presents the EUDI ecosystem of roles, technologies, and standards. Chapter 5 defines a particular use case relating to customer onboarding and the associated identity checks to a Finnish accounting company called Talenom. Chapter 6 explains the ZK-SecreC program that leverages the EUDI framework as it is currently specified to recreate these identity checks in zero-knowledge. Chapter 7 presents some discussion about potential future work and barriers to adopting zero-knowledge technologies. Finally, Chapter 8 provides some concluding remarks about working with ZK-SecreC and the zero-knowledge and digital identity landscapes.

1.2 Verifiable Computation

Protocols for verifiable computation make it possible to verify the correctness of a computation more efficiently than by repeating the computation itself. This enables

outsourcing computation to untrusted parties and has applications in cloud computing and cryptocurrency frameworks. [KPS18]

A public verifiable computation VC consists of a set of three polynomial-time algorithms (`KeyGen, Compute, Verify`) defined as follows: [Par+16]

- $(EK_F, VK_F) \leftarrow \texttt{KeyGen}(F, 1^\lambda)$: The randomized key generation algorithm takes the description of the function F to be outsourced and security parameter λ: it outputs a public evaluation key EK_F and a public verification key VK_F.
- $(y, \pi y) \leftarrow \texttt{Compute}(EK_F, u)$: The deterministic worker algorithm uses the public evaluation key EK_F and input u. It outputs $y \leftarrow F(u)$ and a proof πy of y's correctness.
- $0, 1 \leftarrow \texttt{Verify}(VK_F, u, y, \pi y)$: Given the verification key VK_F, the deterministic verification algorithm outputs 1 if $F(u) = y$, and 0 otherwise.

Related definitions for the notions of correctness, security, and efficiency [Par+16] are given below:

- **Correctness**: For any function F, and any input u to F, if we run $(EK_F, VK_F) \leftarrow \texttt{KeyGen}(F, 1^\lambda)$ and $(y, \pi y) \leftarrow \texttt{Compute}(EK_F, u)$, then we always get $1 = \texttt{Verify}(VK_F, u, y, \pi y)$.
- **Security**: For any function F and any probabilistic polynomial-time adversary A, where $A(EK_F)$ outputs $\hat{u}, \hat{y}$ and $\hat{\pi y}$ and $F(\hat{u}) \neq \hat{y}$, we have that $Pr[\texttt{Verify}((\hat{u}, \hat{y}, \hat{\pi y})) = 1] \leq negl(\lambda)$.
- **Efficiency**: `KeyGen` is assumed to be a one-time operation whose cost is amortized over many calculations, but we require that `Verify` is cheaper than evaluating F.

1.3 Arithmetization

In practical proof systems, the statement to be proven is represented as a set of polynomial equations over a finite field to be solved in a process known as arithmetization. The result of arithmetization can be one of many representations depending on the proof system in question: an arithmetic circuit, a Quadratic Span Program, Rank 1 Constraint System (R1CS), or an Algebraic Intermediate Representation (AIR). [GPR21] An arithmetic circuit is a directed acyclic graph, in which nodes are connected to one another with directed edges. Nodes and edges can also be called gates and wires respectively. The input passes through gates to produce an output. The gates of an arithmetic circuit are field operations such as multiplication or addition. [SML18] For example, Figure 1 shows the circuit that computes the polynomial $(x_1 + x_2)(x_1 + 1)$.

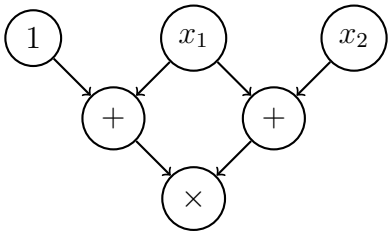


Figure 1. An arithmetic circuit computing the polynomial $(x_1 + x_2)(x_1 + 1)$.

The task of arithmetization is often performed by a compiler, which takes code as input and outputs an executable representation of the equivalent polynomial equations. The compiler can become a bottleneck for zero-knowledge applications, and so its efficiency is a central design concern. Additionally, "only terminating programs can be arithmetized in this way" [SML18], which adds constraints to the higher-level programs that can be compiled.

1.4 Zero-knowledge proofs

Interactive proof systems were presented in [GMR85] and in general consist of a prover and a verifier that operate on shared common input, called the instance, and private input known only to the prover, called the witness. The proof system takes in both inputs and determines whether the witness validates the instance, i.e. whether the statement is true or false. [Bog+22]

More formally, an interactive proof [GMR85] consists of k rounds of communication between a probabilistic polynomial-time (PPT) verifier V and a computationally unbounded prover P. The prover P attempts to prove a statement, i.e. that a word belongs to a language. After the exchange, V either accepts or rejects the statement. Proof systems can be made more robust by increasing the number of rounds k, or by introducing randomness called coins. [Min22] In a private coin protocol, the random choices made by V are kept secret from P, while in a public coin protocol, they are visible to both parties. The proof must satisfy the following properties:

- **Completeness**: Given a statement and a witness, the prover can convince the verifier.
- **Soundness**: A malicious prover cannot convince the verifier of a false statement.

In addition, a proof system may have the property of being zero-knowledge, which means that no other information besides its veracity is revealed in this process. In particular, the prover's witness is not revealed. The proofs produced by zero-knowledge systems are probabilistic, rather than deterministic, meaning that the probability of the existence of a

valid but forged proof is negligibly small. Zero-knowledge proof systems can be further categorised into Succinct Non-Interactive ARguments of Knowledge (zk-SNARKs) and Scalable Transparent ARguments of Knowledge (zk-STARKs). As shown in Table 2, which lists practical implementations of zero-knowledge proof systems, zk-SNARKs are overwhelmingly the protocol of choice in production systems.

A zk-SNARK scheme involves two parties, a prover P and a verifier V, where P proves the correctness of executing a program F on an input $\tilde{x}$ from V, and (optionally) a secret input $\tilde{u}$ from P. P sends V both the output $\tilde{y}$ and a proof $\tilde{\pi}$ to verify the result. Formally, the a SNARK scheme consists of the following three polynomial-time algorithms:

- $(PK_F, VK_F) \leftarrow \texttt{Setup}(F, 1^\lambda)$: given an outsourced program F and a security parameter λ, output a public proving key PK_F and a verification key VK_F. The verification key may be public or private depending on the setting.

- $(\tilde{y}, \tilde{\pi}) \leftarrow \texttt{Prove}(F, PK_F, \tilde{x}, \tilde{u})$: given a program F, the public proving key PK_F, the public input $\tilde{x}$, and the prover's secret input $\tilde{u}$, the output $\tilde{y} \leftarrow F(\tilde{x}, \tilde{u})$, and the proof $\tilde{\pi}$ proving the correctness of the computation.

- $0, 1 \leftarrow \texttt{Verify}(VK_F, \tilde{x}, \tilde{y}, \tilde{\pi})$: given the verification key VK_F, the proof $\tilde{\pi}$, and the statement $(\tilde{x}, \tilde{y})$, output 1 iff $\tilde{y} = F(\tilde{x}, \tilde{u})$.

zk-SNARKs are non-interactive proof systems with small proof size and efficient proving speed. They "support general computations, i.e., they can be used to prove the correctness of arbitrary, polynomial-sized computation" [KPS18] zk-SNARKs require a setup phase to generate initial parameters for the proof process, which is carried out by trusted parties.

zk-STARKs were introduced by Eli Ben-Sasson et al. [Ben+18] in 2018. The term transparency refers to public verifiability without the need for a trusted setup, which mitigates the corruption or attacks than can occur during this phase with zk-SNARKs. Scalability means that proving time scales quasilinearly and verification time scales poly-logarithmically with respect to the size of the witness. [Min22] Transparency means that all messages sent by the verifier are public random coins. zk-STARKs are considered plausibly quantum-secure, because they rely on either the existence of a family of collision-resistant hash functions or common access to a random function. [Ben18] These characteristics make zk-STARKs an attractive object of research for zero-knowledge systems. However, the proof size of zk-SNARKs is approximately 1000 times shorter than that of zk-STARKs, making them a more viable choice in practical systems. Shortening proof size, or aggregating and compressing proofs using incrementally verifiable computation is a open problem in the development of zk-STARKs. [Ben+18]

While zk-SNARKs and zk-STARKs are popular proving systems, according to [Bau+20], their memory constraints and prover runtimes mean they cannot support proofs of very large statements consisting of billions of instructions. Secure multi-party

computation (MPC) protocols are an alternative that can also be used to produce zero-knowledge proofs with bandwidth costs that scale linearly with circuit size. In secure MPC, mutually distrusting parties perform joint computations on private data. In general, the system consists of an n-party functionality $(y_1, ...y_n) = f(x_1, ..., x_n)$ in which player P_i knows x_i. A secure MPC protocol Π for f allows the n parties $(P_1, ..., P_n)$ to learn the outputs such that player P_i learns y_i and nothing more. [CD05] Depending on the security measures taken, Π may also be secure against certain collaborations between some parties in $P_1, ..., P_n$ or against parties who deviate from the protocol. Two-party computation is an instance of multi-party computation where $n = 2$. Zero-knowledge proof protocols can be constructed as two-party computation protocols which are secure against parties deviating from the instructions of the protocol and optimized for the case where one of the parties (the prover) knows all the inputs.

1.5 Concise Binary Object Representation (CBOR)

CBOR is a data serialization format which is defined in the Internet Standard RFC 8949 [BH20]. The encoding of CBOR is inspired by the popular JSON format and is similarly self-descriptive. The 3 higher-order bits in the initial byte contain the major type. The major types are listed in Table 1. Additional information is contained in the 5 lower-order bits.

Major Type	**Meaning**	**Content**
0	unsigned integer N	-
1	negative integer -1-N	-
2	byte string	N bytes
3	text string	N bytes (UTF-8 text)
4	array	N data items (elements)
5	map	2N data items (key/value pairs)
6	tag of number N	1 data item
7	simple/float	-

Table 1. The seven major CBOR types as specified in [BH20].

System	Publication Year	Protocol
Pinocchio [Par+16]	2013	zk-SNARK
TinyRAM [Ben+13]	2013	zk-SNARK
vnTinyRAM [Ben+14]	2014	zk-SNARK
Buffet [Wah+15]	2015	zk-SNARK
Geppetto [Cos+15]	2015	zk-SNARK
Groth16 [Gro16]	2016	zk-SNARK
EMP Toolkit [WMK16]	2016	zk-SNARK
ZKBoo [GMO16]	2016	zk-STARK
Ligero [Ame+17]	2017	zk-SNARK
BulletProof [Bün+18]	2018	Bulletproofs
Hyrax [Wah+18]	2018	zk-SNARK
vRAM [Zha+18]	2018	zk-SNARG
ZoKrates [ET18]	2018	zk-SNARK
xJsnark [KPS18]	2018	zk-SNARK
libSTARK [Ben18]	2018	zk-STARK
OpenZKP [Blo19]	2019	zk-STARK
Aurora [Ben+19a]	2019	zk-SNARK
zk-STARK [Ben+19b]	2019	zk-STARK
Fractal [COS19]	2019	zk-SNARK
Halo [BGH19]	2019	zk-SNARK
Sonic [Mal+19]	2019	zk-SNARK
Marlin [Chi+20]	2019	zk-SNARK
PLONK [GWC19]	2019	zk-SNARK
Hodor [Lab19]	2019	zk-STARK
Virgo [Zha+19]	2020	zk-SNARK
genSTARK [Gui20]	2020	zk-STARK
SuperSonic [BFS20]	2020	zk-SNARK
MIRAGE [Kos+20]	2020	zk-SNARK
Spartan [Set20]	2020	zk-SNARK
Zilch [MT21]	2021	zk-STARK
Nova [KST21]	2021	zk-SNARK
Halo 2 [ZCa22]	2022	zk-SNARK
Winterfell [Fac23]	2023	zk-STARK

Table 2. An overview of practical zero-knowledge proof systems. * Research project.

2 Related Work

In this chapter, we explore other zero-knowledge domain-specific languages (DSLs) and describe their applications, which are primarily in blockchain technologies. Languages developed for verifiable computation are closely related to ZK-SecreC, and they are also described here. We also explain the SIEVE program, a coordinated initiative to develop different components of producing zero-knowledge proofs for large statements.

2.1 Zero-knowledge DSLs

This section provides an overview of domain-specific DSLs designed for zero-knowledge applications. Many DSLs are designed for use with commercial blockchain technologies, highlighting support for writing smart contracts and distributed applications. Many of these languages are imperative and some are developed for use with a specific blockchain, such as Ethereum, Mina, or Aleo.

Cairo is an open-source language and framework to create provable programs for general programming. Various supplementary tools are available, such as a compiler from the Cairo language to Cairo byte code and a virtual machine for simulating Cairo executions. [Sta23] Cairo is developed by StarkWare and designed for use with Ethereum. The idea is to produce an off-chain prover to process large computations, such as large batches of transactions, and produce exponentially smaller validity proofs, which can be verified on-chain. The proofs produced by Cairo are STARKs. [Sta20b] Cairo has a Turing-complete von Neumann architecture which defines a set of universal polynomial equations which can represent the execution of an arbitrary computer program written for some fixed instruction set. [GPR21] Thus, a single verifier can be used to verify any program written in Cairo.

Noir is an open-source Rust-based language for writing zero-knowledge circuits and programs. Noir compiles to an intermediate language called ACIR (Abstract Circuit Intermediate Representation), which can then be compiled to an arithmetic circuit or an R1CS, making it compatible with various proving systems such as "Aztec Brettenberg, Turbo Plonk, and potential future integrations like Groth16 and Halo2". [Azt23] Noir is developed by Aztec and is designed for integration with the Aztec blockchain network. Noir offers direct compilation of verifiers into Solidity smart contracts.

o1js is a TypeScript framework for writing zero-knowledge applications developed by O(1) Labs. `o1js` is an evolution of `SnarkyJS` and designed to work with Mina, a recursively verifiable blockchain. [Lab23b] Mina uses zk-SNARKs for verification,

resulting in a compact blockchain which is around 22kb in total size. [Lab24] The Mina Protocol supports zero-knowledge smart contracts written in `o1js`.

Leo is a functional, statically typed high-level language, which is designed for writing private applications for the Aleo blockchain. [Wu23] Leo is complemented by a toolkit which includes a testing framework, package registry, import resolver, remote compiler, and theorem generator. Leo programs are compiled to intermediate Aleo instructions, which can be further compiled to byte code to be executed by the Aleo Virtual Machine (AVM). Programs are formally verified upon compilation.

Circom is a language designed for zero-knowledge circuit development and developed by iden3. [Bay23] It has been used in several real-world applications such as Dark Forest and Tornado Cash. The `circom` compiler, which is written in Rust, compiles circuits written in the `circom` language. Compatible proving systems developed by iden3 are available in various packages: `SnarkJS` in JavaScript and pure Web Assembly, `wasmsnark` in native Web Assembly, and `rapidSnark` in C++ and Intel Assembly. The CircomLib library contains templates for implementing standard functions such as comparators, but the language is otherwise low-level and designed for writing custom zero-knowledge circuits.

Lurk is a functional programming language for recursive zk-SNARKs. It is a statically scoped Lisp dialect. A Rust implementation called `lurk-rs` generates binaries with `rustc` and supports the full range of features available in Lurk: expression evaluation, proof of correct evaluation, and proof verification. A Common Lisp reference implementation, `lurk`, provides a language specification and expression evaluation. Lurk is designed to integrate with various backend proving systems, such as Groth16 with SnarkPack+ and Nova, or Halo2 in theory. According to [Lab23a], "[t]he most interesting uses of Lurk are those which require proofs of evaluation. A Lurk proof may verifiably attest that X evaluates to Y while revealing no other information."

2.2 Verifiable Computation

Snårkl ("Snorkel") is an open-source Haskell DSL for verifiable computation which supports features found in functional languages, such as sums, products, user-defined inductive datatypes, and case analysis. In Stewart, Merten, and Leland [SML18], constraint minimization, as applied to the arithmetic encodings of programs written in Snårkl, is shown to generate small circuits with low proving and key-generation times. The result of the Snårkl compiler shown in Figure 2 is an R1CS, which is compatible with the `libsnark` backend.

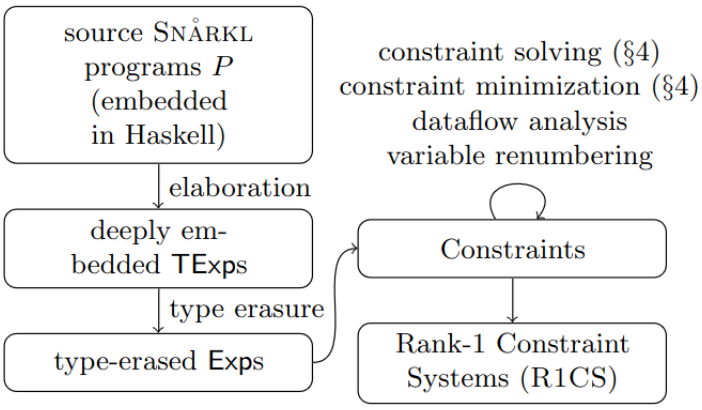


Figure 2. A diagram of the Snårkl compiler as described in [SML18].

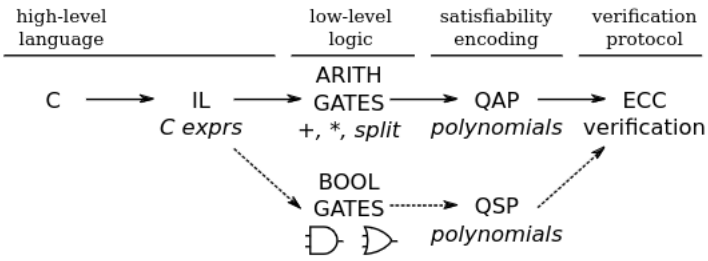


Figure 3. The Pinocchio toolchain as described in Parno et al. [Par+16]. The system transforms a high-level C program into a distributed set of executables that run a verifiable program.

Pinocchio is a system for verifying general computations, with intended applications in outsourced or distributed computing. The system [Par+16] describes a worker and a client, which are analogous to a prover and verifier respectively. The client chooses the input data and desired function, and the worker performs the computation. In order to verify the computation, the client creates a public evaluation key, which the worker can use to produce a signature of computation. This signature is then publicly verifiable using a verification key also created by the client. The system also supports zero-knowledge verifiable computation, in which the worker convinces the client that it knows some input without revealing any information about the input. The Pinocchio toolchain shown in Figure 3 compiles a subset of C into programs that implement the verifiable computation protocol.

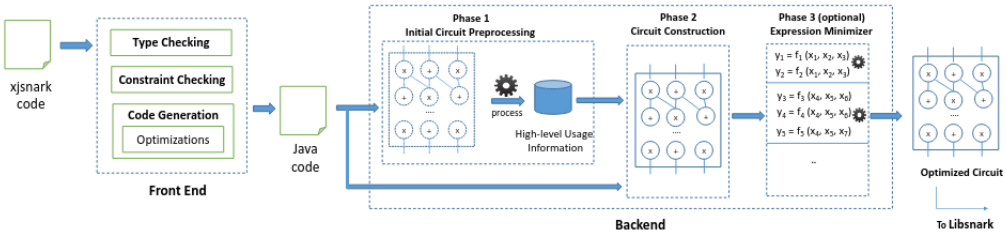


Figure 4. Overview of the xJsnark framework as defined by Kosba, Papamanthou, and Shi [KPS18].

xJsnark is a programming framework for verifiable computation using SNARKs. [KPS18] The framework aims to combine accessibility to programmers with circuit optimizability. This is achieved by an approach in which the source language and compile-time optimizations are co-designed with the goal of minimizing the compiled circuit. The contributions by Kosba, Papamanthou, and Shi [KPS18] include circuit-friendly algorithms for frequent operations such as memory accesses and short and long integer arithmetic which can be expressed as a compact arithmetic circuit with a minimal number of multiplication gates. The xJsnark framework emits circuits compatible with `libsnark`. Figure 4 shows an overview of the xJsnark system.

2.3 SIEVE Program

The DARPA initiative for Securing Information for Encrypted Verification and Evaluation (SIEVE) [Wal] is a program for advancing the state of the art for zero-knowledge, with the specific goal of verifying large, complex statements in a plausibly post-quantum secure way. The program is an international effort to coordinate solving various sub-problems of the overall objective: developing a language to express statements, an encoding of statements into intermediate representations, and protocols to conduct proofs between the prover and verifier. [Bar23]

SIEVE Circuit IR The SIEVE Circuit intermediate representation (IR) is a standardized format for encoding zero-knowledge proof statements produced by a zero-knowledge frontend as arithmetic or boolean circuits, which can then be used as input for a zero-knowledge backend as shown in Fig 5. The SIEVE IR specification [Bun+22] defines two isomorphic formats: text, which is readable by humans, and binary, for automated use. Two backends that currently integrate with the SIEVE IR are the EMP Toolkit and Mac'N'Cheese. They implement two-party secure computation protocols that have been optimized to the case where the prover knows all the inputs.

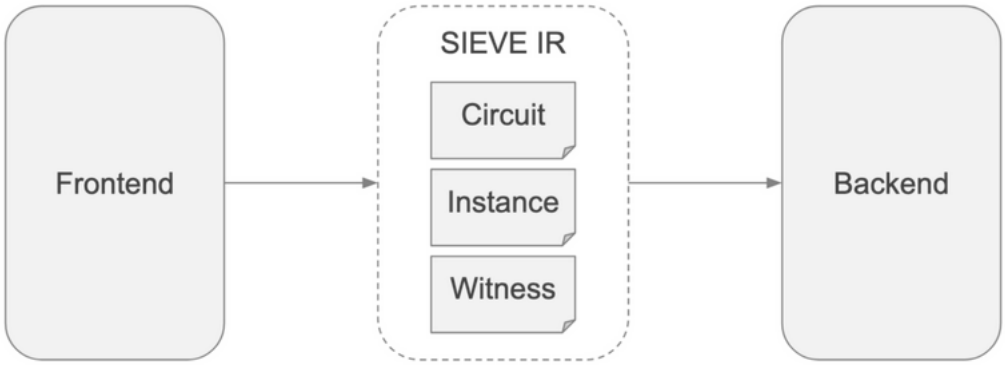


Figure 5. The SIEVE IR is a standard interface between circuit compilers and proving suites from [Gal23]

The EMP Toolkit The EMP Toolkit [WMK16] can be used to perform secure multi-party computation (MPC) in a research and development setting. The toolkit includes the EMP-zk tool, which provides zero-knowledge proof protocols for Boolean and arithmetic circuits and polynomials. The zero-knowledge protocols in the suite include QuickSilver [Yan+21], Mystique [Wen+21], and Wolverine [Wen+20]. The EMP compiler translates the SIEVE IR into its native C++ dialect. It can also interface directly with the ZK-SecreC compiler.

Mac'N'Cheese Mac'N'Cheese [Bau+20] is an interactive proof system developed by Galois, Inc. for boolean and arithmetic circuits which is designed to support large circuits. The system uses the commit-and-prove paradigm using Message authentication Codes (MACs). It supports efficient proving over a network and can also be made non-interactive. The integration of the ZK-SecreC compiler with Mac'N'Cheese supports multithreading, preallocation of wires, and variable buffer size.

3 ZK-SecreC

In this chapter, we describe the ZK-SecreC language and present the features that will be relevant later in the thesis. A program from the documentation [Cyb24] specifying a circuit to prove that a prover knows the cube root of a number is described.

3.1 General

ZK-SecreC is a functional DSL for creating zero-knowledge proofs developed by Cybernetica and published in 2022 [Bog+22]. In contrast to other zero-knowledge DSLs, many of which are designed specifically for use with blockchain, the use cases envisioned for ZK-SecreC are existing industrial applications that may benefit from a zero-knowledge integration. [Reb23] In such applications, the proof sizes are very large and the structure of input data may be complex. In order to support the scope of these potential use cases, the language provides a high degree of granularity in defining where computations are performed (on or off the circuit) and which parties the computations are visible to.

ZK-SecreC is strongly typed, which ensures that non-supported operations cannot be added to the arithmetic circuit. The ZK-SecreC type system also supports the free interleaving of local (off-circuit) computations with computations on the circuit. The language defines two stages and three domains to specify the availability of values to different parties on or off the circuit. The stage `$pre` denotes that a value is available for local computations by the prover and verifier, while `$post` indicates its availability in the circuit. The domain `@prover` denotes that only the prover knows the value, `@verifier` that it is known to both prover and verifier, and `@public` that the value is known at compile time.

The ZK-SecreC compiler takes as input a program written in the ZK-SecreC language, public parameters, witness and instance. The latter three input files must contain exactly one JSON object that represents all input values as strings or nested arrays of strings. [Nes22] Integer inputs must be represented as strings (`52` → `"52"`) and string inputs must be encoded as arrays of the numeric values of the byte encoding of the string (`"test"` → `0x74657374` → `["116", "101", "115", "116"]`). Byte arrays can be similarly represented as strings of the numeric values of the bytes. The front-end of the ZK-SecreC compiler is implemented "in about 20kLoC of Haskell code" [Bog+22], which includes a lexer, parser, type and effect checker, `@public` precomputation engine, and a translater to a simpler intermediate language called Core ZK-SecreC. The back-end of the compiler, which is written in Rust, compiles Core ZK-SecreC to produce a SIEVE Circuit IR. The prover can use the same circuit description to convince the verifier about the witnesses for several different instances.

3.2 Cube Root

ZK-SecreC programs have the extension .zksc. The following program determines whether the prover knows the cube root of a number:

```
type N : Nat = 0x1FFFFFFFFFFFFFFF;

fn cube(x : uint[N] $post @prover) -> uint[N] $post $@prover {
    let a = 1;
    let b = a * x;
    let c = b * x;
    let d = c * x;
    d
}

fn main() where Field[N] {
    let z = wire { get_instance("z") };
    let x = wire { get_witness("x") };
    let y = cube(x);
    assert_zero(y - (z as @prover));
}
```

The first line defines the type-level natural number `N`, which is the 9th Mersenne prime $2^{61} - 1$ in hexadecimal notation. ZK-SecreC requires the number to be a prime, and the programmer must ensure primality. The function `cube` takes takes one parameter, `x`, which has the type constraint `uint[N]`. This means it must be an element of a finite field of size `N`. In addition, it must be in the stage `$post` i.e. on the circuit and in the domain `@prover` i.e. known only to the prover. The function cubes the input and returns the result. An alternative way to program this would be to use the `mut` keyword to create a mutable variable.

```
fn cube(x : uint[N] $post @prover) -> uint[N] $post $@prover {
    let mut result = 1;
    result = result * x * x * x;
    result
}
```

In both cases, the stage and domain of the result do not need to be explicitly declared because the compiler uses the constraints on the input and output of the function to infer them. Functions can also be polymorphic

```
fn cube[N: Nat, $S, @D](x : uint[N] $S @D) -> uint[N] $S $@D {
    let mut result = 1;
    result = result * x * x * x;
    result
}
```

in which case the function can be used with values in different stages, domains, and moduli. Branching is also supported, so that different functionality is executed depending on the stage or domain.

The function `main()` is the entry point for the program. The keyword `where` indicates a `Field` type constraint for the function, and declares that `N` satisfies this constraint. The

functions `get_instance` and `get_witness` fetch fields from the input JSON files for instance and witness. These functions produce results in the `$pre` stage. In order to lift them to the circuit, the values are wrapped in a `wire` construct, which creates an input wire to the circuit and changes the stage of the value to `$post`. The value `x` is cubed and the result `y` is compared with the value `z` from the instance. Because `y` and `z` are in different domains, z must be lifted to the prover's domain (`as @prover`). Finally, the `assert_zero` function asserts that the difference between two inputs is zero. The following contents can be found in the witness file

```
{ "x": "101" }
```

and the instance file

```
{ "z": "1030301" }
```

respectively. To compile the program, we run the command

```
./runrust cube.zksc '' cube_instance.json cube_witness.json -o cube
```

which takes as arguments paths to the public, instance, and witness files. Since there is no public input, the first argument is an empty string. The `-o` flag indicates what the output files should be named. The three files that comprise the SIEVE IR are produced: the circuit in `cube.rel`

```
version 2.1.0;
circuit;
@type field 2305843009213693951;
@begin
  $0 <- @public(0);
  $1 <- @private(0);
  $2 <- @mulc(0: $1, <1>);
  $3 <- @mul(0: $2, $1);
  $4 <- @mul(0: $3, $1);
  $5 <- @mulc(0: $0, <2305843009213693950>);
  $6 <- @add(0: $4, $5);
  @assert_zero(0: $6);
@end
```

the prover's input stream in `cube_0.wit`

```
version 2.1.0;
private_input;
@type field 2305843009213693951;
@begin
  <101>;
@end
```

and the verifier's input stream in `cube_0.ins`.

```
version 2.1.0;
public_input;
@type field 2305843009213693951;
@begin
    <1030301>;
@end
```

The first line declares the version of the SIEVE IR and the second indicates which of the three files is in question. When running a zero-knowledge protocol on this data, the prover has access to the circuit and both input streams, and the verifier only has access to the `cube.rel` and `cube_0.ins` files. Although these two files are always used together, they are differentiated to reflect the conceptual difference between the relation and public input. Some backends may also perform preprocessing that only uses the `.rel` file, such as creating a relation-specific common reference string (CRS) with a trusted setup.

ZK-SecreC provides various libraries for standard building blocks for programming such as Integer, String, Float, Date etc. The standard toolkit includes support for cryptographic constructs like SHA-256 and elliptic curves. For real-world applications, functionality for parsing data in different input formats like JSON and CBOR have also been built. Another such library is Challenges, which implements verifier challenges. These random challenges are issued after the prover has committed to some part of the witness. The prover must then construct the remainder of the witness so that the proof will be accepted. A common application of verifier challenges is in proving that two committed lists are permutations of one another. These libraries are available for import in ZK-SecreC programs and used in our application.

4 The European Union Digital Identity (EUDI) Framework

This chapter provides an overview of the technologies and standards within the EUDI framework. The roles in the architecture and the state of EUDI development in Finland are also described.

4.1 General

The European Commission adopted a recommendation [Bre] in June of 2021 to develop a Union Toolbox, which includes the European Digital Identity Architecture and Reference Framework (ARF) [Exp22]. The ARF is a document that defines a set of standards, technical specifications, best practices and common guidelines for interoperable EUDI wallet technologies. The ARF may also encompass functionality or security recommendations that are not mandatory for compliance, but a subset of the results of the Toolbox development will form the basis upon which solutions developed by member states are accepted into the actual EUDI ecosystem. The Toolbox is an open collaboration between teams representing the interests of member states, which can be followed on GitHub. [De +] Two privacy guarantees to take into consideration with respect to architectural choices are selective disclosure and unlinkability.

Selective disclosure refers broadly to the possibility to share a minimum required subset of attributes or attestations instead of an entire credential. For example, when only age is required, the relying party does not learn any other attributes about the user when checking their credential. Unlinkability is a privacy guarantee regarding the meta-data of the credential. By monitoring the circumstances under which a credential is used, a profile of the user's behavior and attributes can be built. Unlinkability refers to the amount of data that colluding verifiers, who in this case are relying parties, can infer about a user. [ETS23] For example, although a user has shared their data selectively with different verifiers, it may be possible for them to use a correlating factor such as a public key to synchronize the data they have each obtained about a user. Collusion with issuers is also possible. High unlinkability means that little can be learned about the user through collusion.

4.2 Standards

The standard credential formats selected for the Type 1 or primary configuration of the wallet are the ISO mobile driving license (mDL) and the W3C Verifiable Credentials with SD-JWT. These standards were chosen because they are mature and have been approved by the Senior Officials Group for Information Systems (SOG-IS), which is a requirement

for use with EUDI. [ETS23] The ARF also specifies a Type 2 or secondary configuration which accounts for technologies that are less mature or are pending auditing and approval.

4.2.1 Selective Disclosure - JSON Web Token (SD-JWT)

The draft presented by Fett, Yasuda, and Campbell [FYC24] specifies conventions for selectively disclosing individual elements of a JSON object secured by the JSON Web Signature (JWS). The data in the JWT payload are called claims. They are represented in JSON and digitally signed, so that the recipient can verify that the set of claims have not been modified since being signed. The JWT is encrypted during transit, but an unencrypted JWT can be used to read all signed claims in plaintext. The EUDI ARF, as well as other verifiable credential schemes, call for JWTs that are signed when they are issued, and then re-used with different verifiers. In this case, in order to verify the one claim has not been modified, the verifier would learn the JWT holder's entire set of signed claims. Selective disclosure enables the user to share only a minimum subset of the signed claims required by a verifier to ascertain the integrity of those claims cryptographically.

4.2.2 Mobile Driving License (mDL) / MobileSecurityObject (MSO) Scheme

The ISO/IEC 18013 family of documents [Sta20a] is a standard published in 2020 which specifies the requirements for physical and mobile driver's licenses (mDL) for government use. [Fla24] Part ISO/IEC 18013-5 specifies the `mdoc` format, a structure for mDLs. Two modes of retrieval are specified in the standard: online and offline. In online retrieval, the end user authorises a trusted authority to share an mDL stored on their servers with a relying party. In offline retrieval, the trusted authority issues the mDL to a device, which the user can share directly with relying parties. Because the device is not trusted, the trusted authority issues supplementary MobileSecurityObjects (MSOs), which contain fieldwise hashes of the mDL signed by the issuer. The relying party can use these to verify the integrity of the data supplied by a untrusted device.

In online retrieval, "the trusted authority is present in each transaction" [Sta20a] and can therefore easily monitor the user's behaviour and their use of the mDL. Offline retrieval is decentralized, and as such can be used to facilitate a self-sovereign identity (SSI) scheme, in which users have full control of a credential after it has been issued. Then the issuer cannot monitor the user, and the mDL can be used without access to a centralized authority. The trust model for both modes of retrieval is founded on public key infrastructure (PKI), and "requires a mechanism to distribute and disseminate the set of Certification Authorities certificates from issuing authorities." [Sta20a] The relying party is responsible for checking the status of the issuer certificate. The architectural differences between the two retrieval modes are are compared in Figure 6.

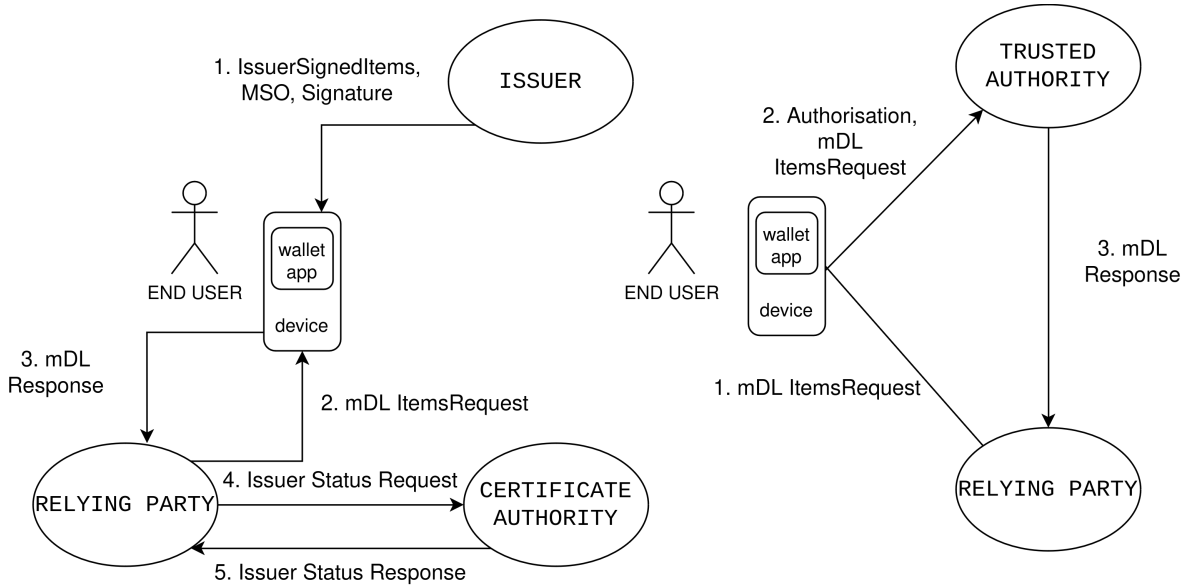


Figure 6. The offline and online retrieval modes in the ISO mDL specification.

4.2.3 Alternative Standards

The selected standards for the Type 1 configuration provide the privacy guarantee of selective disclosure, but they do not directly facilitate exchanging zero-knowledge proofs or using stronger privacy-preserving protocols. Various such protocols have been standardized: AnonCreds by Hyperledger, the BBS Signature Scheme by IETF, BBS Cryptosuite and the Verifiable Credentials Data Model by W3C. [ETS23] Discussion on the EUDI ARF GitHub acknowledges that these protocols would provide strong privacy guarantees, but identifies barriers to integration: lack of widespread hardware support and architectural incompatibility. [GSM] The zero-knowledge toolchain presented in this thesis differs from these alternative standards in that ZK-SecreC programs can be tailored to the existing mDL data structures. Only a protocol communicating zero-knowledge proofs about a credential would need to be standardized. This would lower the threshold to access privacy-preserving credentials under EUDI by minimizing the amount of architectural changes required.

4.3 The EUDI Ecosystem

The EUDI ARF [Exp22] defines the various roles in the ecosystem, which are summarized below. The same organisation may assume several different roles in a given member state. The dependencies between the roles are illustrated in Figure 7.

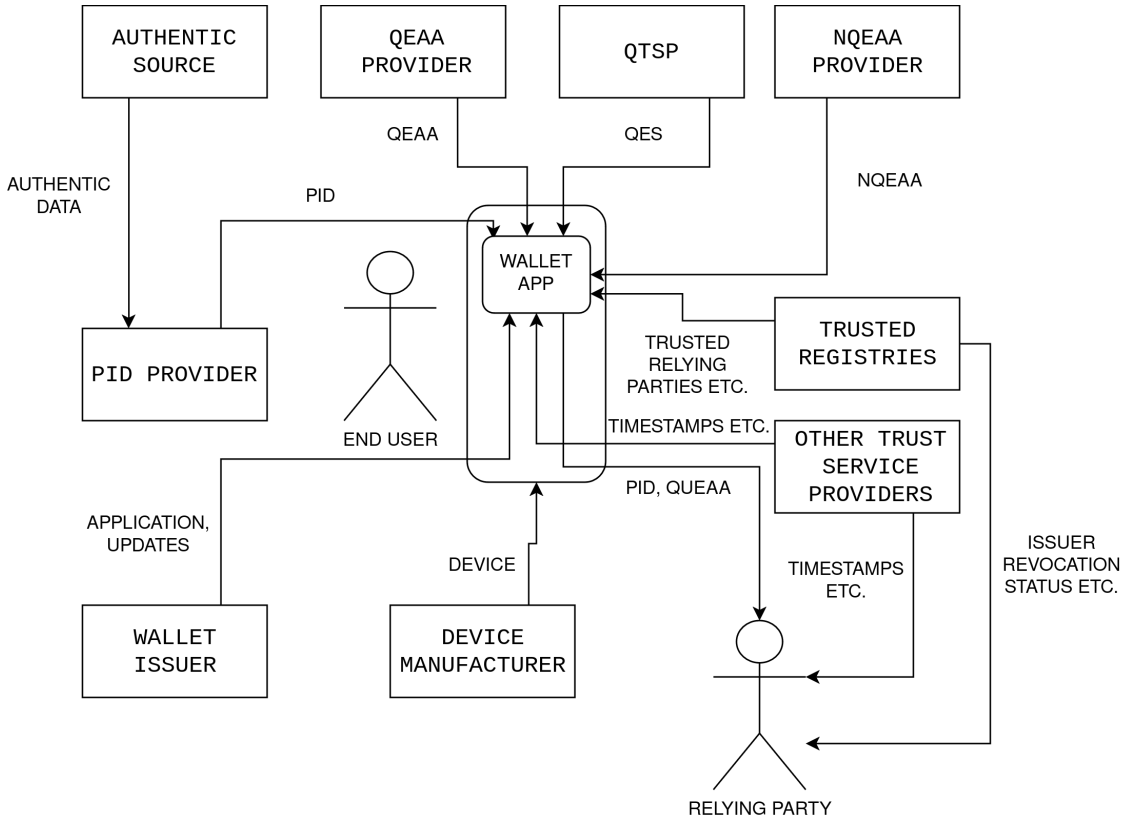


Figure 7. A diagram of the different roles in the EUDI ecosystem, excluding CABs and supervisory bodies, which supervise the whole system.

End users are legal or natural person who uses the wallet to receive, store, and share personal identification data (PID) and attestations (QEAA, EAA).

Wallet issuers are organisations mandated or recognized by member states to make wallets available to end users. Wallet issuers are responsible for the wallet complying with the functional, non-functional, and security requirements described in the ARF.

Personal Identification Data (PID) providers are responsible for verifying the identities of end users and maintain a standardized interface to issue PIDs to wallets. They also maintain information needed by relying parties to verify the validity of PIDs such as revocation lists and public keys.

Qualified Electronic Attestation of Attributes (QEAA) providers are Qualified Trust Service Providers (QTSPs) that interface with authentic sources and wallets to provide attestations, such as electronic signatures, regarding the veracity of PIDs.

Non-Qualified Electronic Attestation of Attributes (NQEAA) providers can provide various trust services, such as attestations for professional qualifications. Such attestations would have to be technically compliant with EUDI in order to be stored in the wallet, but may be subject to legal and contractual frameworks other than eIDAS2.

Providers of other trust services such as timestamps or certificates may be components of the ecosystem. These may be qualified or non-qualified as demanded by the use case.

Authentic sources are public or private repositories containing valid data about the attributes of legal or natural persons. This data may include names, birth date, civil status, and gender, while other repositories may contain information about education, professional qualifications, or permits and licenses.

Relying parties are natural or legal persons who request electronic attestations or attributes in the PID dataset from end users. Their reliance on the wallets of end users may be a consequence of a legal requirement, a contractual obligation, or their own decision. In mutual authentication, relying parties may themselves share attestations or attributes. Relying parties are responsible for the authentication of the attestations or attributes they receive.

Conformity Assessment Bodies (CABs) are accredited public or private bodies that perform regular audits of providers in the ecosystem. They are responsible for conferring the qualified status to EAA providers and ensuring compliance of other providers being introduced to the ecosystem.

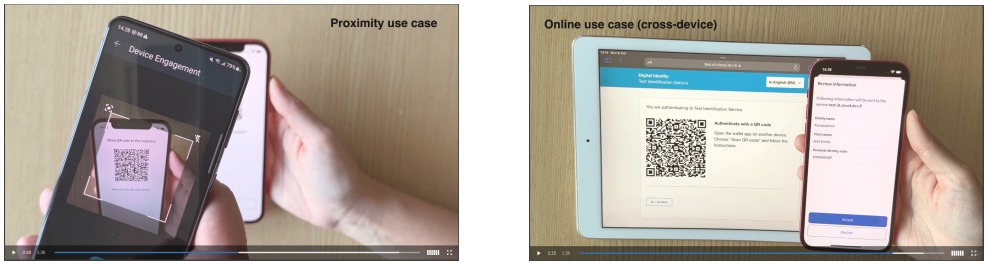
Device manufacturers and related subsystem providers refer to all device manufacturers and service providers underpinning the ecosystem: manufacturers of mobile hardware, operating systems, and sensors, developers of networking protocols, cryptographic algorithms, and secure hardware, and cloud storage and app store providers, etc. Which devices and services comply with EUDI security requirements will be defined by the ARF.

Supervisory bodies supervise QTSPs and NQEAA providers.

Trusted registries may maintain trusted registries of any of the above roles. They may be responsible for processing applications and admitting new parties into trusted registries and providing real-time information on their trust status. The management of such registries represents an open problem in the architecture, depending on the demanded scope. A registry of relying parties, for example, could number in the tens of millions.

4.4 EUDI in Finland

The Digital and Population Data Services Agency of Finland (DVV) is responsible for national co-operation with the eIDAS2 Toolbox process. This entails implementing compliant pilot applications and ensuring the interests of the country, such as offline usage, are represented in the development of the ARF. The Finnish EUDI Wallet Demo enables testing various use cases in the EUDI framework. [PL] The application references the open-source identity-android wallet application developed by the Open Wallet Foundation for Android. [Ope] This wallet application matches the ISO mDL / MSO specification and uses QR codes to encode requests and responses. In the proximity or in-person use case, a PID or QEAA approved by the user is presented to an mDL reader application on another device. In the online or cross-device use case, the wallet application reads a request for PIDs or QEAAs from another device, such as a laptop. These two use cases are demonstrated in Figure 8. In both use cases, the pilot application assumes offline retrieval according to the ISO mDL specification in order to minimize reliance on trusted authorities.



(a) Proximity use case.

(b) Cross-device use case.

Figure 8. Screenshots from the demo video of the EUDI wallet application developed by DVV. [PL]

5 Defining the Use Case: User Onboarding

In this section, we scope out a potential use case under the future EUDI framework. The use case concerns Talenom, a Finnish accounting company that also provides third-party financial services, such as accounting, payment, and banking services. The primary customers of the company are businesses, which are represented by persons who have a mandate to act on behalf of the business. In order to onboard a new customer, Talenom must obtain the following assurances:

- user's age is over 18
- user's nationality is EU
- user possesses mandate to act on behalf of the organisation in question

First we describe how these checks are implemented under the existing digital identity infrastructure in Finland, and then we propose how these assurances might be obtained with a zero-knowledge extension of EUDI.

5.1 Current Strong Authentication Flow

Strong authentication in Finland is facilitated by the Finnish Trust Network (FTN), a national registry of identification service providers that is managed by the National Cyber Security Centre Finland (NCSC-FI). The NCSC-FI is a department of Traficom, the Finnish Transport and Communications Agency. The FTN is a list of service providers that are licensed to provide identification means, identification broker services, or both. [Kyb24] The identification means available in Finland are online banking credentials, mobile certificates issued by teleoperators, and identification certificates stored on smart cards issued by the Digital and Population Data Services Agency (DVV). Identification

broker services aggregate identification means providers and forward identification events to them. Banking credentials are overwhelmingly the most popular means of identification [MSV19], making up 95.3% of identification events recorded by major identification broker service Suomi.fi in 2018, followed by mobile certificates at 4.1% and other methods at 0.6%.

eIDAS defines three levels of assurance, which indicate "the degree of confidence in the claimed identity" [Com24]. Identity solutions are classified as either Low, Substantial, or High assurance based on a security assessment, number of authentication factors, and integrity of the enrolment process. The level of assurance of the most popular identity means in Finland is classified as Substantial, because customers are required to present a valid identity document in person in order to open a bank account or phone contract. Authentication with these credentials requires at least two of the following factors: knowledge-based (username and password), inherent (biometric), or ephemeral (one-time) factors.

Provider	**Type**	**Service**	**Assurance**
Aktia bank P.L.C.	Bank	Means	Substantial
Danske Bank A/S	Bank	Means, Broker	Substantial
DNA PLC	Teleoperator	Means, Broker*	Substantial
Elisa Corporation	Teleoperator	Means, Broker	Substantial
Nets Branch Norway	Commercial Trust	Broker	High
Nordea Bank ABP	Bank	Means, Broker	Substantial
Oma Savings Bank PLC	Bank	Means	Substantial
OP Cooperative †	Bank	Means, Broker	Substantial
POP Bank Group	Bank	Means	Substantial
Signicat AS	Commercial Trust	Broker	Substantial
S-Bank Ltd	Bank	Means	Substantial
Svenska Handelsbanken AB	Bank	Means	Substantial
Säästöpankkiryhmä	Bank	Means	Substantial
Telia Finland Oyj	Teleoperator	Means, Broker	Substantial
Population Register Centre (DVV)	Agency	Means	High
Bank of Åland PLC	Bank	Means	Substantial
Assently AB	Commercial Trust	Broker	Substantial
Megical Oy	Commercial Trust	Means, Broker	Substantial

Table 3. Registered identification services in Finland. [Kyb24] † Provides Suomi.fi broker service for public services. * For other teleoperators only.

The typical strong authentication flow under the current system is shown in Figure 9. The customer is redirected to a portal hosted by an information broker service, such as the one shown in Figure 10. The customer selects their identity means provider, and is

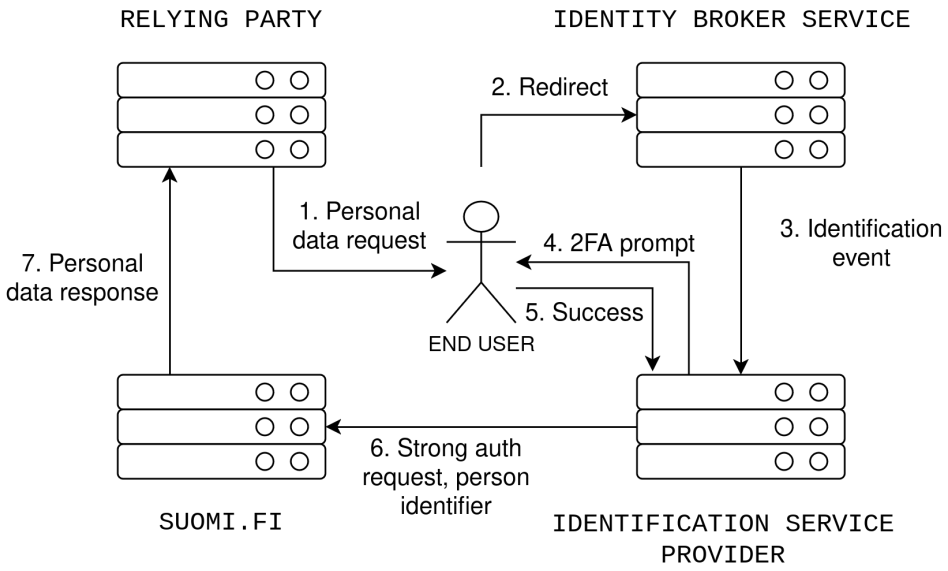


Figure 9. Current strong authentication flow with broker, identification means providers, and Suomi.fi API.

prompted to perform two-factor authentication with their online bank credentials, mobile phone, or smart card. The identification means providers interface with Suomi.fi, an online Web API that facilitates access to the following national databases and registers [Suo23]:

- personal data in the Population Information System [Digital and Population Data Services Agency]
- system of environmental division of real estate [National Land Survey of Finland]
- trade register [National Board of Patents and Registration of Finland]
- registers of associations and religious communities [National Board of Patents and Registration of Finland]
- vehicle data and watercrafts [Traficom]
- driving licence data [Traficom]
- check your own employment pension institution [The Finnish Centre for Pensions]
- right to study and education information [Finnish National Board of Education]

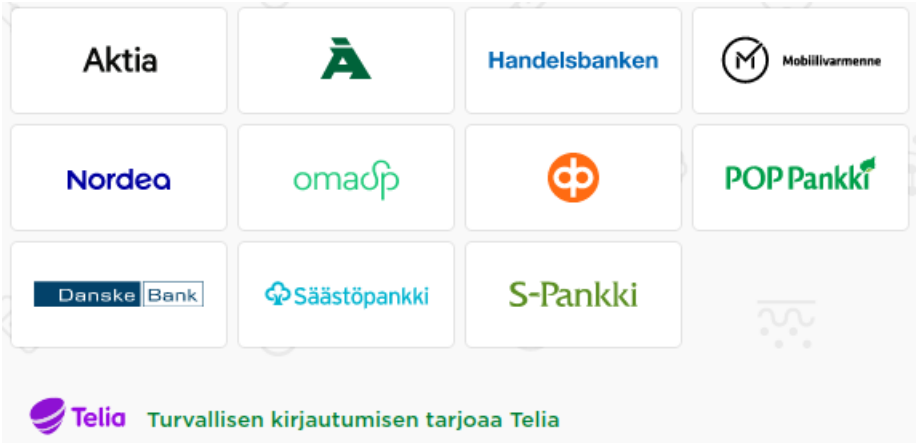


Figure 10. A typical view from an identification broker service in Finland. This particular broker service is the teleoperator Telia.

Data requested by a third party from any of the above are delivered back with a JWT along with a status message that describes the success of the authentication. For some use cases, such as login, only the successful status message from the strong authentication is required, and any other data returned by Suomi.fi can be discarded. In other use cases, such as customer onboarding, the FTN is used to fetch data from the national database, such as the personal identity code, name, nationality, or address, which may be used to perform due diligence checks. These are determined by Know-Your-Customer (KYC) legislation and best practices, and include checking that the customer's nationality is not considered "high-risk", that their name is not subject to sanctions, or that their permanent place of residence is in Finland.

Suomi.fi e-Authorisations is a service which enables a relying party to "verify the mandates of a person or a company to act online on behalf of another person or a company they represent" [Suo24]. Here, mandates are fetched from a national registry. Different API endpoints allow either fetching all mandates or checking a specific mandate. Often, the user selects which organisation they want to share their mandate for from a menu in the browser. In some cases, a person does not have a mandate to act alone on behalf of a organisation, but has a mandate to act together with another person. Synchronising authentication between several parties for this use case is an open problem.

5.2 Target Zero-Knowledge Flow with EUDI

The components for this use case are the presentation protocol, the ZK-SecreC compiler and programs, and the proving backend. In accordance with the goal of making zero-knowledge technologies available inside the existing EUDI framework, the presentation protocol should leverage the existing issuing architecture and format of credentials issued

to the wallet. The CDDL structure of an mDL Reader Request sent by a relying party is the following:

```
OfflineRequest = {
"version" : tstr, ;
"docRequests" : [+ DocRequest] // Requested DocType, NameSpace and data elements.
}
DocRequest = {
"itemsRequest" : ItemsRequestBytes,
? "readerAuth" : ReaderAuth
}
ItemsRequestBytes = #6.24(bstr .cbor ItemsRequest)
ReaderAuth = COSE_Sign1
ItemsRequest = {
? "docType" : DocType,
"nameSpaces" : NameSpaces,
? "requestInfo" : {* tstr => any} // Additional info the reader wants to provide.
}
DocType = tstr
NameSpaces = {
+ NameSpace => DataElements // Requested data elements for each NameSpace.
}
NameSpace = tstr
DataElements = {
+ DataElement => IntentToRetain
DataElement = tstr
IntentToRetain = bool
```

The `docType` defines the type of document being requested or returned. For the ISO mDL, the `docType` is `org.iso.18013.5.1.mDL`. The `nameSpace` specifies a definition for the data elements within the document. Different countries and issuing authorities are expected to add their own namespaces for defining local data representations. The recommended naming convention for new namespaces appends the ISO 3166-1 alpha-2 country code or the ISO 3166-2 region code to the mDL namespace i.e. `org.iso.18013.5.1.EU-FI`. One document can accommodate a request for several different namespaces. For requesting and returning zero-knowledge proofs, a new `docType` would need to be defined. For our use case, the relying party requests zero knowledge assurances for their age, EU citizenship, and a mandate for acting on behalf of the organization with business code "1234567-8". The address and port to connect to to execute the proving backend could be communicated in the optional `requestInfo` field. A deserialized `ItemsRequest` might then look like the following:

```
{
    "docType": "org.iso.zk.1",
    "nameSpaces":
    {
        "org.iso.zk.1.EU-FI":
        {
            "is_over_18": true,
            "is_citizen": "EU",
            "has_mandate": "1234567-8"
        },
    }
    "requestInfo": "192.168.0.0:30000"
}
```

A component in the wallet application or a separate user application could then interpret this request and interface with the mDL by sending a request of its own for the items it needs for these assurances:

```
{
    "docType": "org.iso.18013.5.1.mDL",
    "nameSpaces":
    {
        "org.iso.18013.5.1.EU-FI":
        {
            "birth_date": true,
            "issue_date": true,
            "expiry_date": true,
            "nationality": true,
            "mandates": true
        }
    }
}
```

Here we assume that the local Finnish namespace defines a field that contains a list of mandates to act on behalf of an organisation. The `OfflineResponse` in the mDL specification has the following structure:

```
OfflineResponse = {
    "version" : tstr,
    ? "documents" : [+Documents],
}
Documents = {
    + DocType => ResponseData
}
DocType = tstr
ResponseData = {
    "issuerSigned" : IssuerSigned, // Responded data elements signed by the issuer.
}
IssuerSigned = {
    ? "nameSpaces" : IssuerNameSpaces,
    "issuerAuth" : IssuerAuth
}
IssuerAuth = COSE_Sign1 // The payload is the MobileSecurityObject.
IssuerNameSpaces = {
    + NameSpace => [ + IssuerSignedItemBytes ]
}
IssuerSignedItemBytes = #6.24(bstr .cbor IssuerSignedItem)
NameSpace = tstr
IssuerSignedItem = {
    "digestID" : uint // Dynamic identifier to correlate with value digests in MSO.
    "random" : bstr // 16-byte salt for value digest.
    "elementIdentifier" : tstr // Field name as shown in Appendix.
    "elementValue" : any
}
```

The array of serialized `IssuerSignedItem` structures is what we obtain as secret input for the ZK-SecreC circuit. The MSO structure contains hashes of each `IssuerSignedItem` signed by the issuing authority. This will be the verifier's input for the circuit. Examples of these two structures are described in more detail in Section 6.1. In our imagined communication protocol, the prover would then return the `.rel` and `.ins` files produced

by the compiler and the MSO to the verifier. The response could have the following structure:

```
{
    "docType": "org.iso.zk.1",
    "responseData": {
    "nameSpaces":
        {
            "issuerSigned": {
                "org.iso.zk.1.EU-FI":
                {
                    ".rel": 0x123, // CBOR serialization of .rel
                    ".ins": 0x123, // CBOR serialization of .ins
                },
                "issuerAuth": Cose_Sign1 // Signed hashes of the used items.
            }
        }
    }
}
```

The relying party can then check the signature on the MSO and ensure that the verifier input stream in the `.ins` file contains the signed hashes. A compatible proving backend such as Mac'N'Cheese can then be executed over the network on to obtain the requested assurances. Figure 11 demonstrates the target zero-knowledge flow with EUDI.

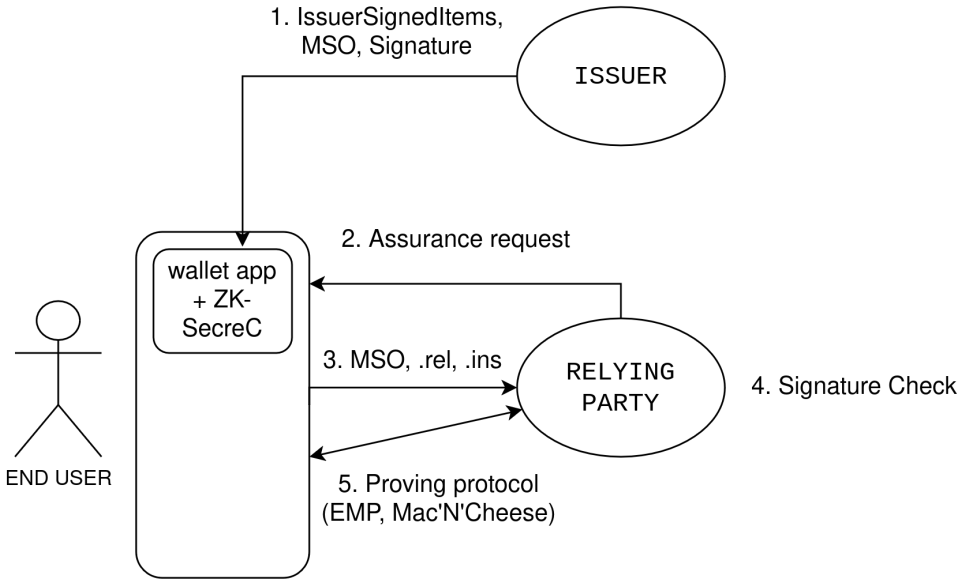


Figure 11. The flow for presenting zero-knowledge credentials within the EUDI framework.

6 Applying ZK-SecreC to User Onboarding

In this section, we explore the ZK-SecreC program that implements the functionality described in Section 5. We also describe verifying this program with the EMP Toolkit and Mac'N'Cheese and present some performance metrics on the program.

6.1 Inputs

It was established in Section 5.2 that five fields in the mDL are required for performing our checks: `birth_date`, `issue_date`, `expiry_date`, `nationality`, and `mandates`. The three CBOR types for these fields are tagged date, string, and an array of strings. The `mandates` field contains a list of business codes the credential holder has the right to act on behalf of. Although we assume here that the mandate list is a field in the mDL, it could also be a completely separate credential under EUDI. `DigestIDs` for these items are chosen randomly between 1 and 100. This results in the following `IssuerSignedItem` objects:

```
[
    {
        "digestID": 93,
        "random": 0x48d7eed5bf382f1566e6df48610b81e9,
        "elementIdentifier": "birth_date",
        "elementValue": CBORTag(18013, "06-06-1998")
    },
    {
        "digestID": 77,
        "random": 0xfc219632fd0c4cd9488f1891d152e004,
        "elementIdentifier": "issue_date",
        "elementValue": CBORTag(18013, "01-01-2024")
    },
    {
        "digestID": 85,
        "random": 0x85aaee0944928ab9f890de2c5a3a3217,
        "elementIdentifier": "expiry_date",
        "elementValue": CBORTag(18013, "12-12-2024")
    },
    {
        "digestID": 21,
        "random": 0x611d3485bb478fd4143100188302f74a,
        "elementIdentifier": "nationality",
        "elementValue": "FI"
    },
    {
        "digestID": 35,
        "random": 0x05835315a86dddfe1d64d2f65f17637b,
        "elementIdentifier": "mandates",
        "elementValue": ["1234567-8", "7654321-8"]
    },
]
```

The associated MSO has the following structure, with irrelevant fields omitted:

```
MobileSecurityObject = {
    "digestAlgorithm" : tstr,
    "valueDigests" : ValueDigests, // Array of digests of all data elements.
    "validityInfo" : ValidityInfo
}
ValueDigests = {
    "nameSpaces" : NameSpacesDigests
}
NameSpacesDigests = {
    + NameSpace => DigestIDs
}
DigestIDs = {
    + DigestID => Digest
}
ValidityInfo = {
    "signed" : tdate,
    "validFrom" : tdate,
    "validUntil" : tdate,
}
NameSpace = tstr // NameSpace as used in IssuerSigned.
DigestID = uint // DigestID as used in IssuerSignedItem.
Digest = bstr // digest(IssuerSignedItem)
```

The MSO corresponding to the above IssuerSignedItems:

```
MobileSecurityObject = {
    "digestAlgorithm": "SHA-256",
    "valueDigests": {
        "nameSpaces": {
            "org.iso.18013.5.1": {
                93: "7924993d7ee9b47be4985899bae5b6ba8eb4e43b32c42ce08107b35dc2b277eb",
                77: "4f93e0b17309c979e88f5ee6e002ec4bf5f99b989d2c330ce4576c01959c6b45",
                85: "cf87cb6bfdbadf147c5d15502da677ee864c0edf3fad552a6a391a050aa871a6",
                21: "925ed3b4d7d8166e638e31946b35bb3241095904d3fa710b6678a43f4271a7e2",
                35: "da787a19101303d8f972190bb18e4db005b1972a56c9fb448337cc66317ac0c4"
            }
        }
    },
    "validityInfo": {
        "signed": CBORTag(18013, '2024-02-01'),
        "validFrom": CBORTag(18013, '2024-02-01'),
        "validUntil": CBORTag(18013, '2024-06-06')
    }
}
```

A Python script was written for pre-formatting inputs according to the assumptions presented in Section 5.2. The `IssuerSignedItems` and `MSO` objects are generated and then serialized in CBOR, and the resulting bytes are represented as numeric strings for input into the circuit. The public parameters include publicly known information like the list of EU countries for comparing nationality to, and the current date. For checking the mDL holder is of age, their `birth_date` must be earlier than or equal to the date 18 years ago. There is no need to compute this date on circuit, so the `born_by` date is provided in the public parameters. The lengths of any variable size arrays (`mso_cbor_len`, `mandates_num`, `lens_issuer_signed_items`) and data

describing the shape of the mDL (`mDL_field_val_lens`, `mDL_field_max_strlens`, `mDL_field_max_sublens`, `mDL_field_max_deps`) are also provided in the public parameters

```
{
    "today_year": "2024",
    "today_month": "5",
    "today_day": "7",
    "eu_countries_num": "27",
    // Each element of the array contains the numeric value of the ASCII bytes
    spelling the ISO country code.
    "eu_countries": [["65", "84"], ["66", "69"], ["66", "71"], ["67", "89"],
    ["67", "90"], ["68", "69"], ["68", "75"], ["69", "69"], ["69", "83"],
    ["70", "73"], ["70", "82"], ["71", "82"], ["72", "82"], ["72", "85"],
    ["73", "69"], ["73", "84"], ["76", "84"], ["76", "85"], ["76", "86"],
    ["77", "84"], ["78", "76"], ["80", "79"], ["80", "84"], ["82", "79"],
    ["83", "69"], ["83", "73"], ["83", "75"]],
    "born_by_year": "2006",
    "born_by_month": "5",
    "born_by_day": "12",
    "issuer_signed_items_len": "5",
    // The bytelengths of the IssuerSignedItems serialized in CBOR.
    "lens_issuer_signed_items": ["92", "92", "93", "81", "97"],
    "mso_cbor_len": "333",
    "mandates_num": "2",
    // Data for configuring the CBOR parser to the shape of the mDL and MSO.
    // See 'Parsing CBOR' below.
    "mDL_field_val_lens": ["10", "10", "10", "9", "11"],
    "mDL_field_max_strlens": ["17", "17", "17", "17", "17"],
    "mDL_field_max_sublens": ["4", "4", "4", "4", "4"],
    "mDL_field_max_deps": ["3", "3", "3", "2", "3"],
    "mso_val_len": "30"
}
```

instance,

```
{
    // Numeric values of the ASCII bytes spelling "1234567-8"
    "mandate": ["49", "50", "51", "52", "53", "54", "55", "45", "56"],
    // Bytes that represent MobileSecurityObject serialization in CBOR.
    "mso_cbor": ["163", "111", "100", ..., "48", "54"]
}
```

and witness.

```
{
    "issuer_signed_items": [
    ["164", "104", ... , "54"], // birth_date IssuerSignedItem bytes in CBOR.
    ["164", "104", ... , "49"], // issue_date IssuerSignedItem bytes in CBOR.
    ["164", "104", ... , "50"], // expiry_date IssuerSignedItem bytes in CBOR.
    ["164", "104", ... , "83"], // nationality IssuerSignedItem bytes in CBOR.
    ["164", "104", ... , "56"] // mandates IssuerSignedItem bytes in CBOR.
    ]
}
```

6.2 Program

Preamble We initialize the circuit with the type-level natural number N61, which is the prime number $2^{61} - 1$ in hexadecimal notation. The number must be a prime, but this particular prime is chosen because the it is supported by the EMP proving backend.

```
type N61 : Nat = 0x1FFFFFFFFFFFFFFF;
```

We also declare the use of the following standard libraries and extra programs `hashes` and `cbor`:

```
use Std::*;
use String::*;
use Date::*;
use Integer::*;
use Inequalities::*;
use hashes::*;
use cbor::*;
```

In ZK-SecreC, `structs` are finite sequences of values of various types. The following `structs` aid extracting the contents of the `IssuerSignedItems` by extending the `Date` and `String` types in the ZK-SecreC `stdlib` with a native `digestID`.

```
struct MDL_date[$S, @D] {
    value: Date[$S, @D, N61],
    digest_id: uint[N61] $S @D
}

struct MDL_string[$S, @D] {
    value: String[$S, @D, N61],
    digest_id: uint[N61] $S @D
}

struct MDL_string_list[$S, @D] {
    value: list[String[$S, @D, N61]],
    digest_id: uint[N61] $S @D
}
```

The `structs` called `MDL` and `MobileSecurityObject` reflect the data structures in the ISO specification:

```
struct MDL[$S, @D] {
    birth_date: MDL_date[$S, @D],
    issue_date: MDL_date[$S, @D],
    expiry_date: MDL_date[$S, @D],
    issuing_country: MDL_string[$S, @D],
    nationality: MDL_string[$S, @D],
    mandates: MDL_string_list[$S, @D]
}

struct ValidityInfo[$S, @D] {
    signed: Date[$S, @D, N61],
    validFrom: Date[$S, @D, N61],
    validUntil: Date[$S, @D, N61]
}

struct MobileSecurityObject[$S, @D] {
    valueDigests: list[list[uint[N61] $S @D]],
    validityInfo: ValidityInfo[$S, @D]
}
```

Parsing CBOR `IssuerSignedItemBytes` and the payload in `COSE_Sign1` are serialized in CBOR. To parse input data in this format, we make use of the CBOR parsing library in ZK-SecreC presented in [Nes22] to extract the contents of the mDL from the set of `IssuerSignedItems`. The CBOR parser provides functions to extract definite-length data of major types 0 and 1 (integers), 2 (byte strings), 3 (text strings), 4 (arrays), 5 (maps), and dates, which are recognized as such by the tag number 18013. This number refers to the ISO mDL specification. According to [Nes22], "zero knowledge does not enable transforming an input in the CBOR format to some delimited representation reflecting its semantic structure since the sizes of subtrees and the number of them are not necessarily known". In order to parse the CBOR, a configuration structure must be initialized with information about the expected shape of the mDL. The functions `extract_MDL` and `extract_MSO` shown below initialize the configuration and extract the desired contents.

```
fn extract_MDL(
    issuer_signed_items: list[list[uint[N61] $post @prover]],
    mandates_num: uint $pre @public)
    -> MDL[$post, @prover] where Field[N61], Challenge[N61] {
    fn extract_MDL(
    issuer_signed_items: list[list[uint[N61] $post @prover]],
    mandates_num: uint $pre @public) -> MDL[$post, @prover] where Field[N61], Challenge[N61] {
    let mDL_fields = length(issuer_signed_items);

    // Values for CBOR configuration. Each value corresponds to an IssuerSignedItem.
    let mDL_field_val_lens : list[uint $pre @public] = get_public("mDL_field_val_lens");
    let mDL_field_max_strlens : list[uint $pre @public] = get_public("mDL_field_max_strlens");
    let mDL_field_max_sublens : list[uint $pre @public] = get_public("mDL_field_max_sublens");
    let mDL_field_max_deps : list[uint $pre @public] = get_public("mDL_field_max_deps");

    // ASCII bytes spelling digestID.
    let digestId_key = [100, 105, 103, 101, 115, 116, 73, 68];
    // ASCII bytes spelling elementValue.
    let elementValue_key = [101, 108, 101, 109, 101, 110, 116, 86, 97, 108, 117, 101];

    // Configure CBORs.
    let mut cbors = for i in 0 .. mDL_fields {
        let config = CborConfig
          {
            total_len: length(issuer_signed_items[i]),
            val_len: mDL_field_val_lens[i], // Number of CBOR headers.
            max_strlen: mDL_field_max_strlens[i], // Maximum string length.
            max_sublen: mDL_field_max_sublens[i], // Maximum number of pairs in map or element in array.
            max_dep: mDL_field_max_deps[i] // Maximal depth of nesting.
          }
        ; let arrays = cbor_init_arrays(config)
        ; let stores = cbor_init_stores()
        ; Cbor { config: config, arrays: arrays, stores: stores }
    };

    // Fetching the values is omitted.

    MDL {
        birth_date: birth_date_mdl_date,
        issue_date: issue_date_mdl_date,
        expiry_date: expiry_date_mdl_date,
```

```
        issuing_country: issuing_country_mdl_string,
        nationality: nationality_mdl_string,
        mandates: mandates_mdl_string_list
    }
}
```

The array of `CborConfig` structures describe each `IssuerSignedItem`. Therefore, the ordering of items must be consistent, or a proof cannot be successfully generated. The types of the items can be inferred from the configuration: the maximum depth of nesting for an `IssuerSignedItem` representing a tagged date or arrays is 3, and the maximum depth of nesting for one representing a string is 2. The number of CBOR headers in the `IssuerSignedItem` is 10 for tagged dates, 9 for strings, and $9 + len(array)$ for an array. The maximum number of pairs in an `IssuerSignedItem` is 4 by default, unless the item represents an array of length longer than 4. The maximum length for each of these particular `IssuerSignedItems` is 17, the length of the longest string `'elementIdentifier'`. The length of strings can be obscured by setting some upper limit rather than the actual maximum length of a string in the item, but this is less efficient.

```
fn extract_MSO(
    mso_cbor: list[uint[N61] $post @prover],
    mdl: MDL[$post, @prover]
    ) -> MobileSecurityObject[$post, @prover] where Field[N61], Challenge[N61] {
    let mso_val_len : uint $pre @public = get_public("mso_val_len");
    // Configure CBOR.
    let config = CborConfig {
        total_len: length(mso_cbor),
        val_len: mso_val_len, // Number of CBOR headers in MSO.
        max_strlen: 32, // SHA-256 length.
        max_sublen: length(issuer_signed_items), // Maximum number of elements in array or pairs in map.
        max_dep: 5 // Maximal depth of nesting.
    };

    // Fetching values is omitted.

    MobileSecurityObject {
        valueDigests: value_digests_val,
        validityInfo: validity_info
    }
}
```

Integrity Checks Checking the integrity of the supplied mDL entails asserting that the fieldwise hashes match those provided in the MSO. This is performed by the function `check_integrity()` shown below.

```
fn check_integrity(
        mso : MobileSecurityObject[$post, @prover],
        digest_ids : list[uint[N61] $post @prover],
        issuer_signed_items : list[list[uint[N61] $post @prover] $pre @public] $pre @public,
    ) where Field[N61] {
    for i in 0 .. length(digest_ids) {
        // Check that IssuerSignedItem hashes match MSO valueDigests.
```

```
        check_hash(issuer_signed_items[i], mso.valueDigests[i]);
    };
}
```

Validity Checks Verification of the issuer's signature and validation of the issuer's certificate can happen off-circuit. On-circuit checks include validating that the current date is between the `valid_from` and `valid_until` fields in the MSO and that the validity period is between the `issue_date` and `expiry_date` in the mDL. This function takes as one of its parameters `ref sizeasserter`. A `SizeAsserter` is a mutable object that can be used to check that a binary representation of a number fits into a fixed number of bits. The `SizeAsserter` object is initialized in `main()` with the number of bits given as an argument. This is then given to `check_validity` as a pass-by-reference indicated by the keyword `ref`.

```
fn check_validity(
        mdl : MDL[$post, @prover],
        mso : MobileSecurityObject[$post, @prover],
        ref sizeasserter : SizeAsserter[N61, $post, @prover]
    ) where Field[N61] {
    let today_year = wire { get_public("today_year") };
    let today_month = wire { get_public("today_month") };
    let today_day = wire { get_public("today_day") };
    let today_date : Date[$post, @public, N61] = date(today_year, today_month, today_day);

    // The timestamp of 'validFrom' shall be equal or later than the 'signed' element.
    date_assert_le(mso.validityInfo.signed, mso.validityInfo.validFrom, ref sizeasserter);

    // The value of the validUntil timestamp shall be later than the 'validFrom' element.
    date_assert_le(mso.validityInfo.validFrom, mso.validityInfo.validUntil, ref sizeasserter);

    // The 'validFrom' timestamp shall not be before the value of the 'issue_date' element from the mDL.
    date_assert_le(mdl.issue_date.value, mso.validityInfo.validFrom, ref sizeasserter);

    // The 'validUntil' timestamp shall not be beyond the value of the 'expiry_date' element from the mDL.
    date_assert_le(mso.validityInfo.validUntil, mdl.expiry_date.value, ref sizeasserter);

    // The 'validFrom' timestamp is before or equal to today.
    date_assert_le(mso.validityInfo.validFrom, date_to_prover(today_date), ref sizeasserter);

    // Today is before or equal to the 'validUntil' timestamp.
    date_assert_le(date_to_prover(today_date), mso.validityInfo.validUntil, ref sizeasserter);
}
```

Assurances The function `assurances()` checks the three specific use case requirements:

```
fn assurances(
        mdl: MDL[$post, @prover],
        ref sizeasserter : SizeAsserter[N61, $post, @prover]
    ) where Field[N61] {
    // Nationality must be in the EU.
    let eu_countries_num : uint $pre @public = get_public("eu_countries_num");
    let eu_countries_pre : list[list[uint[N61] $pre @public]] = get_public("eu_countries");
    let eu_countries : list[list[uint[N61] $post @public]] = for i in 0 .. eu_countries_num {
```

```
        for j in 0 .. 2 { wire { eu_countries_pre[i][j] } }
    };
    let eu_countries_strs : list[String[$post, @public, N61]] = for i in 0 .. eu_countries_num {
        String { chars: eu_countries[i], len: 2 }
    };

    let mut is_eu_citizen = false;
    for i in 0 .. eu_countries_num { is_eu_citizen = is_eu_citizen |
        string_eq(mdl.nationality.value, string_to_prover(eu_countries_strs[i]), 2)
    };

    // Age must be over 18.
    let born_by_year = wire { get_public("born_by_year") };
    let born_by_month = wire { get_public("born_by_month") };
    let born_by_day = wire { get_public("born_by_day") };
    let born_by_date : Date[$post, @public, N61] = date(born_by_year, born_by_month, born_by_day);

    date_assert_le(mdl.birth_date.value, date_to_prover(born_by_date), ref sizeasserter);

    // Must possess specified mandate.
    let mandate_pre : list[uint[N61] $pre @verifier] $pre @verifier = get_instance("mandate");
    let mandate_chars = for i in 0 .. 9 { wire { mandate_pre[i as @verifier] } };
    let mandate : String[$post, @verifier, N61] = String { chars: mandate_chars, len: 9 };

    let mandates_num : uint $pre @public = get_public("mandates_num");
    let mut has_mandate = false;
    for i in 0 .. mandates_num { has_mandate = has_mandate |
        string_eq(string_to_prover(mandate), mdl.mandates.value[i], 9)
    };

    assert(is_eu_citizen & has_mandate)
}
```

6.3 Verification

EMP Toolkit Running the EMP proving backend with the produced program required some changes to be made to the program. As the EMP Toolkit is for research and development, the EMP backend does not support verifier challenges, which the cbor library functions utilise. This was solved by using permutation networks instead of verifier challenges for the underlying sorting requirements of the CBOR parser by importing and using the standard `Waksman` library. This has the disadvantage of being slower than using verifier challenges. Running the compiler with the EMP flag replaces the SIEVE IR generation with calls to the `EMP-zk` tool through a wrapper. The compiler takes as the first four arguments paths to the program, public parameters, instance, and witness respectively. The additional circuits argument loads the SHA-256 circuit. The prover and verifier are run concurrently with the following commands

```
./runrust mdl_checker.zksc mdl_checker_public.json mdl_instance mdl_checker_instance.json \\
mdl_checker_witness.json circuits --emp --prover

./runrust mdl_checker.zksc mdl_checker_public.json mdl_instance mdl_checker_instance.json \\
mdl_checker_witness.json circuits --emp --verifier
```

to obtain a result.

Mac'N'Cheese The Mac'N'Cheese backend supports verifier challenges, and so it was possible to run the program without modifications. Setting the `-mnc` flag runs the ZK-SecreC compiler with the Mac'N'Cheese integration. The Mac'N'Cheese amount of threads, and the amount of threads per field can be set with the flags `-threads` and `-threads-per-field` respectively. The `-preallocate` flag can be used to preallocate a specific number of wires for the circuit, which is 200 000 000 by default. The `-buffer-size` flag defines how often the prover and verifier must communicate. The buffer size is 200 000 by default. Similarly to the EMP Toolkit, the prover and verifier are run concurrently with the following commands

```
./runrust mdl_checker.zksc mdl_checker_public.json mdl_checker_instance.json \\
mdl_checker_witness.json --mnc --addr 127.0.0.1:30000 --preallocate 20000 \\
--buffer-size 2000 --threads 7 --threads-per-field 2

./runrust mdl_checker.zksc mdl_checker_public.json mdl_checker_instance.json \\
mdl_checker_witness.json --mnc --verifier --addr 127.0.0.1:30000 --preallocate 20000 \\
--buffer-size 2000 --threads 7 --threads-per-field 2
```

6.4 Performance

Flamegraphs can be used to visualize the stack traces of software. Profiling the Rust code generated by the compiler using a flamegraph utility can provide insight into which parts of the program are the most resource-intensive. The functions `extract_MDL` and `extract_MSO` have the deepest stack traces, while the most resource-intensive function is `check_integrity`. This indicates that the number of fields in the mDL, and therefore the amount of hashes that need to be checked, represents a potential bottleneck for the program.

Since the cost of the other functions in the application are minimal, a ZK-SecreC program for benchmarking the combination of CBOR parsing and integrity checks was written. Here we skip the `assurances()` and `check_validity()` functions. For simplicity, we assume all fields are strings, and so each `IssuerSignedItem` can use the same `CborConfig`. Since we only need the `digestID` to correlate the `IssuerSignedItems` with the `ValueDigests` in the MSO, we need not extract the actual element values of the fields. A new function `extract_digest_ids()` was written for this purpose:

```
fn extract_digest_ids(
    issuer_signed_items: list[list[uint[N61] $post @prover]]
    ) -> list[uint[N61] $post @prover] where Field[N61], Challenge[N61] {
    let mDL_fields = length(issuer_signed_items);
    // ASCII bytes spelling digestID.
    let digestId_key = [100, 105, 103, 101, 115, 116, 73, 68];

    // Configure CBORs.
    let mut cbors : list[Cbor[N61, $post, @prover]] = for i in 0 .. mDL_fields {
        let config = CborConfig
          {
            total_len: length(issuer_signed_items[i]),
            val_len: 9, // Number of CBOR headers for string type IssuerSignedItem.
```

No of Fields	Compilation	-buffer-size 200000	-buffer-size 2000
5	72.9s	4.1s	8.1s
10	73.5s	6.1s	12.6s
15	73.7s	8.5s	16.3s
30	74.1s	14.8s	29.6s
50	74.3s	24.7s	45.1s
70	75.2s	34.6s	63.1s
100	77.7s	47.3s	89.0s

Table 4. Effect of number of fields and -buffer-size flag on runtime.

```
            max_strlen: 17, // Maximum string length from 'elementIdentifier'.
            max_sublen: 4, // Maximum number of pairs in string type IssuerSignedItem map.
            max_dep: 2 // Maximal depth of nesting in string type IssuerSignedItem.
          }
        ; let arrays = cbor_init_arrays(config)
        ; let stores = cbor_init_stores()
        ; Cbor { config: config, arrays: arrays, stores: stores }
    };

    // Initialize CBORs.
    let mut large_sizeasserter = sizeasserter_new(log2(N61)-2);
    let mut small_sizeasserter = sizeasserter_new(5);
    let mut medium_sizeasserter = sizeasserter_new(50);

    for i in 0 .. length(issuer_signed_items) {
        cbor_init(
            issuer_signed_items[i],
            ref cbors[i],
            ref large_sizeasserter,
            ref small_sizeasserter,
            ref medium_sizeasserter
        );
    };

    // Get digestId pointers.
    let digest_id_vals : list[uint[N61] $post @prover] = for i in 0 .. mDL_fields {
        let digest_id_ptr = cbor_lookup_map_strkey_with_check(ref cbors[i], 0, digestId_key);
        cbor_get_val(ref cbors[i], digest_id_ptr)
    };

    digest_id_vals
}
```

The functions extract_mso() and check_integrity() remain unchanged. A Python program was written to generate different sizes of input data, i.e. arrays of IssuerSignedItems for the program. The program was benchmarked on a laptop with an Intel i5-8350U CPU from 2017. The results of running Mac'N'Cheese with default options on different datasets are shown in Table 4. The significance of network latency was demonstrated by limiting communication between the prover and verifier using the -buffer-size flag.

6.5 Security Properties

In summary, using the described zero-knowledge protocol provides the relying party with strong assurances that the attributes of the prover fulfil their requirements. The authenticity and validity of the credentials used to generate proofs stem from the underlying EUDI framework, which facilitates issuing credentials by trusted authorities. By definition, the zero-knowledge proof reveals nothing about else about the statement to be proved. In particular, the relying party does not learn the prover's age, nationality, or list of mandates. The relying party does learn the shape of the mDL and the length of the mandate list in particular, as these are public. The shape of the mDL includes the ordering, types, and maximum lengths of the `IssuerSignedItems`.

7 Discussion

In this chapter, we explore the future work to be done to integrate with the mDL standard and other potential applications related to our use case. Other legislative and social challenges to adopting zero-knowledge technologies are also addressed.

7.1 Future Work

Further functionality demanded by the ISO mDL specification includes adding support for two other digest algorithms, SHA-384 and SHA-512, and support for the `tdate` format, as the `ValidityInfo` in the MSO is provided in `tdate` format. Support for the full range of CBOR types would complete the zero-knowledge CBOR parsing library. The results from benchmarking indicate a strong correlation between the number of fields in the mDL and runtime, with one field adding approximately 0.4 seconds to the runtime with the default buffer size. Since the number of fields in the mDL can reach up to over a hundered fields [Sta20a], verifying the entire mDL would take between one and several minutes depending on network latency. The efficiency of hashing is then an important area of development for this use case. Another approach would be to utilise the support for polymorphism over moduli in ZK-SecreC and Mac'N'Cheese and compute hashes $\bmod 2$ and other checks $\bmod N61$. Using the SD-JWT standard to expose a minimum subset of items which still have a valid signature could also be used to minimize the amount of hashing required for integrity checks.

7.2 Other Challenges

In an open letter [Ope23] to the European Parliament, critics of eIDAS2 question why the regulation only enables, rather than mandates, the use of privacy-preserving technologies and unlinkability guarantees in the EUDI Wallet. Given the requirements for an interoperable architecture, the member state that implements the weakest security guarantees for its citizens will determine the available protections for all other EU citizens. Service providers, such as Google or Facebook, can exploit this loophole by registering in the member state with the protections most amenable to their business. Thus, the development of strong technical protections risks being undermined by legislation that does not explicitly mandate that they be used.

Another potential barrier to accessing the privacy benefits of privacy-preserving technologies with eIDAS2 for EU citizens is the lack of broad adoption of the new infrastructure. In Finland, most public and private services have already been brought online. The use of online banking credentials to authenticate to these services is ubiquitous and familiar to the majority of citizens. According to a 2019 report by the Finnish Ministry of Finance [MSV19], the "Substantial" level of assurance provided by these identification means is sufficient and even preferable to the "High" alternative due to cost-efficiency, as

development and maintenance costs have been outsourced to commercial identification means providers. These factors result in low motivation on part of relying parties to add support for novel authentication methods. This is a problem even for adopting the mDL standard, and support for a zero-knowledge namespace could be even harder to motivate.

8 Conclusion

This thesis explored the potential of using zero-knowledge technology within the EUDI framework in its current state. A potential use case for this technology in the Finnish digital identity landscape was identified and a ZK-SecreC program to perform the required checks was produced by adapting an existing implementation for parsing CBOR. A test suite was constructed for benchmarking the program under different conditions, which indicated that the number of mDL fields represents a bottleneck for the verification program. Since the mDL can have up to 30 fields, optimizing the hash checking is instrumental to ZK-SecreC being adopted for the use case described in this thesis. For ZK-SecreC to be integrated with the EUDI ecosystem generally, protocols for communicating zero-knowledge proof requests and responses must be standardized, and ZK-SecreC should be developed either within EUDI wallet applications or as a separate user application.

As someone new to both zero-knowledge and functional programming, getting accustomed to programming in ZK-SecreC took some time. In addition to the usual constraints in a statically typed language, dealing with stages and domains presented an extra layer of complexity. Exploring the unique language features of ZK-SecreC by writing a "simple" program from scratch proved difficult. The language is both flexible, in that it is possible to interleave `$pre` and `$post` computations and cast between domains, and strict, in that this is hard to do correctly. As a beginner, it felt easier to write programs that relied on standard library functions, as type inference from the underlying functions laid the foundation for what could be done with the program, and resulted in more straightforward errors. Programming was greatly aided by the Visual Studio Code extension, which provided syntax highlighting and error descriptions. Example code and documentation were also very helpful in understanding how to use the language.

The fields of zero-knowledge technologies and digital identity solutions are both evolving rapidly, with notable changes taking place throughout the process of writing this thesis. ZK-SecreC is under active development to achieve open-sourcing by autumn 2024. Development goals include making more use of different Rust native types other than BigInt in the compiler front-end, and the possibility to directly call Rust functions if all values are in the `$pre` stage. Other privacy-preserving standards such as BBS+ are also undergoing approval by cryptography boards. The EUDI pilot projects and discussion on the EUDI ARF development forum are ongoing [Eurb], and a new EUDI Wallet core library project was initialized on GitHub [Eura] in March of 2024. Development is challenging with so many areas of the problem in flux, but the opportunity to offer privacy-preserving technologies to EU citizens is open.

References

[Ame+17] Scott Ames et al. "Ligero: Lightweight Sublinear Arguments Without a Trusted Setup". In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. CCS '17. Dallas, Texas, USA: Association for Computing Machinery, 2017, pp. 2087–2104. ISBN: 9781450349468. DOI: 10.1145/3133956.3134104. URL: https://doi.org/10.1145/3133956.3134104.

[Asz] Alan Aszepieniec. *Anatomy of a SNARK*. URL: https://aszepieniec.github.io/stark-anatomy/.

[Azt23] Aztec. *Noir Documentation*. Tech. rep. Aztec, 2023. URL: https://noir-lang.org/docs/.

[Bar23] Joshua Baron. *I was told there would be blockchain: 5 Years of Real World Crypto at DARPA*. Talk given at Real World Cryptography Conference. 2023. URL: https://rwc.iacr.org/2023/program.php.

[Bau+20] Carsten Baum et al. *Mac'n'Cheese: Zero-Knowledge Proofs for Boolean and Arithmetic Circuits with Nested Disjunctions*. Cryptology ePrint Archive, Paper 2020/1410. 2020. URL: https://eprint.iacr.org/2020/1410.

[Bay23] Jordi Baylina. *Circom Documentation*. Tech. rep. iden3, 2023. URL: https://docs.circom.io/.

[Ben+13] Eli Ben-Sasson et al. *SNARKs for C: Verifying Program Executions Succinctly and in Zero Knowledge*. Cryptology ePrint Archive, Paper 2013/507. 2013. URL: https://eprint.iacr.org/2013/507.

[Ben+14] Eli Ben-Sasson et al. "Succinct non-interactive zero knowledge for a von Neumann architecture". In: *Proceedings of the 23rd USENIX Conference on Security Symposium*. SEC'14. San Diego, CA: USENIX Association, 2014, pp. 781–796. ISBN: 9781931971157.

[Ben+18] Eli Ben-Sasson et al. "Scalable, transparent, and post-quantum secure computational integrity". In: *IACR Cryptol. ePrint Arch.* 2018 (2018), p. 46. URL: https://api.semanticscholar.org/CorpusID:44557939.

[Ben+19a] Eli Ben-sasson et al. "Aurora: Transparent Succinct Arguments for R1CS". In: Apr. 2019, pp. 103–128. ISBN: 978-3-030-17652-5. DOI: 10.1007/978-3-030-17653-2_4.

[Ben+19b] Eli Ben-sasson et al. "Scalable Zero Knowledge with No Trusted Setup". In: Aug. 2019, pp. 701–732. ISBN: 978-3-030-26953-1. DOI: 10.1007/978-3-030-26954-8_23.

[Ben18] Eli Ben-Sasson. *libSTARK: a C++ library for zk-STARK systems*. GitHub. 2018. URL: https://github.com/elibensasson/libSTARK?tab=readme-ov-file.

[BFL90] L. Babai, L. Fortnow, and C. Lund. "Nondeterministic exponential time has two-prover interactive protocols". In: *Proceedings [1990] 31st Annual Symposium on Foundations of Computer Science*. 1990, 16–25 vol.1. DOI: 10.1109/FSCS.1990.89520.

[BFS20] Benedikt Bünz, Ben Fisch, and Alan Szepieniec. "Transparent SNARKs from DARK Compilers". In: May 2020, pp. 677–706. ISBN: 978-3-030-45720-4. DOI: 10.1007/978-3-030-45721-1_24.

[BGH19] Sean Bowe, Jack Grigg, and Daira Hopwood. *Recursive Proof Composition without a Trusted Setup*. Cryptology ePrint Archive, Paper 2019/1021. 2019. URL: https://eprint.iacr.org/2019/1021.

[BH20] C. Bormann and P. Hoffman. *Concise Binary Object Representation (CBOR)*. RFC 8949. RFC Editor, Dec. 2020. URL: https://www.rfc-editor.org/rfc/rfc8949.html.

[Blo19] Remco Bloemen. *OpenZKP - pure Rust implementations of Zero-Knowledge Proof systems.* GitHub. 2019. URL: https://github.com/0xProject/OpenZKP.

[Bog+22] Dan Bogdanov et al. *ZK-SecreC: a Domain-Specific Language for Zero Knowledge Proofs*. 2022. arXiv: 2203.15448 [cs.PL].

[Bre] Thierry Breton. *Commission Recommendation (EU) 2021/946 of 3 June 2021 on a common Union Toolbox for a coordinated approach towards a European Digital Identity Framework*. URL: https://eur-lex.europa.eu/eli/reco/2021/946/oj (visited on 05/15/2024).

[Bün+18] Benedikt Bünz et al. "Bulletproofs: Short Proofs for Confidential Transactions and More". In: *2018 IEEE Symposium on Security and Privacy (SP)*. 2018, pp. 315–334. DOI: 10.1109/SP.2018.00020.

[Bun+22] Paul Bunn et al. *EMP-toolkit: Efficient MultiParty computation toolkit*. 2022. URL: https://raw.githubusercontent.com/sieve-zk/ir/main/v1.0.1/sieve-ir-v1.0.1.pdf.

[CD05] Ronald Cramer and Ivan Damgård. "Multiparty Computation, an Introduction". In: *Contemporary Cryptology*. Basel: Birkhäuser Basel, 2005, pp. 41–87. ISBN: 978-3-7643-7394-8. DOI: 10.1007/3-7643-7394-6_2. URL: https://doi.org/10.1007/3-7643-7394-6_2.

[CGL20] Markus Christen, Bert Gordijn, and Michele Loi. *The Ethics of Cybersecurity*. Springer Cham, 2020. DOI: https://doi.org/10.1007/978-3-030-29053-5.

[Chi+20] Alessandro Chiesa et al. "Marlin: Preprocessing zkSNARKs with Universal and Updatable SRS". In: *Advances in Cryptology – EUROCRYPT 2020*. Ed. by Anne Canteaut and Yuval Ishai. Cham: Springer International Publishing, 2020, pp. 738–768. ISBN: 978-3-030-45721-1.

[Com24] European Commission. *eIDAS Levels of Assurance (LoA)*. 2024. URL: https://ec.europa.eu/digital-building-blocks/sites/display/DIGITAL/eIDAS+Levels+of+Assurance.

[Cos+15] Craig Costello et al. "Geppetto: Versatile Verifiable Computation". English. In: *2015 IEEE Symposium on Security and Privacy*. 2015 IEEE Symposium on Security and Privacy ; Conference date: 18-05-2015 Through 20-05-2015. United States: Institute of Electrical and Electronics Engineers (IEEE), July 2015, pp. 253–270. DOI: 10.1109/SP.2015.23. URL: https://www.ieee-security.org/TC/SP2015/.

[COS19] Alessandro Chiesa, Dev Ojha, and Nicholas Spooner. *Fractal: Post-Quantum and Transparent Recursive Proofs from Holography*. Cryptology ePrint Archive, Paper 2019/1076. 2019. URL: https://eprint.iacr.org/2019/1076.

[Cyb24] Cybernetica. *ZK-SecreC Documentation*. In preparation. 2024.

[De +] Paolo De Rosa et al. *Architecture and Reference Framework*. URL: https://github.com/eu-digital-identity-wallet/eudi-doc-architecture-and-reference-framework/blob/main/docs/arf.md (visited on 05/15/2024).

[dig24] dig.watch. *Digital identities*. 2024. URL: https://dig.watch/topics/digital-identities.

[ET18] Jacob Eberhardt and Stefan Tai. "ZoKrates - Scalable Privacy-Preserving Off-Chain Computations". In: *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*. 2018, pp. 1084–1091. DOI: 10.1109/Cybermatics_2018.2018.00199.

[ETS23] ETSI. *Electronic Signatures and Infrastructures (ESI); Analysis of selective disclosure and zero-knowledge proofs applied to Electronic Attestation of Attributes*. Aug. 2023. URL: https://www.etsi.org/deliver/etsi_tr/119400_119499/119476/01.01.01_60/tr_119476v010101p.pdf.

[EU] EU. *European Digital Identity*. European Commision. URL: https://commission.europa.eu/strategy-and-policy/priorities-2019-2024/europe-fit-digital-age/european-digital-identity_en.

[EU14] EU. *REGULATION (EU) No 910/2014 OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL*. EUR-Lex. 2014. URL: https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=uriserv:OJ.L_.2014.257.01.0073.01.ENG.

[EU21] EU. *Proposal for a REGULATION OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL amending Regulation (EU) No 910/2014 as regards establishing a framework for a European Digital Identity*. EUR-Lex. 2021. URL: https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX%5C%3A52021PC0281.

[Eura] European Digital Identity. *EUDI Wallet Core library for Android*. URL: https://github.com/eu-digital-identity-wallet/eudi-lib-android-wallet-core (visited on 05/15/2024).

[Eurb] European Digital Identity. *EUDI Wallet Repository Activity*. URL: https://github.com/eu-digital-identity-wallet/eudi-doc-architecture-and-reference-framework/pulse (visited on 05/15/2024).

[Exp22] eIDAS Expert Group. *European Digital Identity Architecture and Reference Framework – Outline –*. 2022. URL: https://digital-strategy.ec.europa.eu/en/library/european-digital-identity-architecture-and-reference-framework-outline.

[Fac23] Facebook. *Winterfell*. GitHub. 2023. URL: https://github.com/facebook/winterfell.

[Fla24] Heather Flanagan. *Verifiable Credentials and mdocs – a tale of two protocols*. Spehrical Cow Consulting. 2024. URL: https://sphericalcowconsulting.com/2024/01/03/verifiable-credentials-and-mdocs-a-tale-of-two-protocols/.

[FYC24] D. Fett, K. Yasuda, and B. Campbell. *Selective Disclosure for JWTs (SD-JWT)*. Tech. rep. IETF, 2024. URL: https://www.ietf.org/archive/id/draft-ietf-oauth-selective-disclosure-jwt-08.html.

[Gal23] Inc. Galois. *SIEVE Circuit IR Version 2.1.0 Release*. 2023. URL: https://galois.com/news/sieve-circuit-ir-version-2-1-0-release/.

[GMO16] Irene Giacomelli, Jesper Madsen, and Claudio Orlandi. *ZKBoo: Faster Zero-Knowledge for Boolean Circuits*. Cryptology ePrint Archive, Paper 2016/163. 2016. URL: https://eprint.iacr.org/2016/163.

[GMR85] S Goldwasser, S Micali, and C Rackoff. "The knowledge complexity of interactive proof-systems". In: *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*. STOC '85. Providence, Rhode Island, USA: Association for Computing Machinery, 1985, pp. 291–304. ISBN: 0897911512. DOI: 10.1145/22145.22178. URL: https://doi.org/10.1145/22145.22178.

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. "The Knowledge Complexity of Interactive Proof Systems". In: *SIAM Journal on Computing* 18.1 (1989), pp. 186–208. DOI: 10.1137/0218012. eprint: https://doi.org/10.1137/0218012.

[GPR21] Lior Goldberg, Shahar Papini, and Michael Riabzev. *Cairo – a Turing-complete STARK-friendly CPU architecture*. Cryptology ePrint Archive, Paper 2021/1063. 2021. URL: https://eprint.iacr.org/2021/1063.

[Gro16] Jens Groth. "On the Size of Pairing-Based Non-interactive Arguments". In: *IACR Cryptol. ePrint Arch.* 2016 (2016), p. 260. URL: https://api.semanticscholar.org/CorpusID:2254747.

[GSM] GSMA Europe. *Privacy shall be at the heart of ARF*. URL: https://github.com/eu-digital-identity-wallet/eudi-doc-architecture-and-reference-framework/issues/66 (visited on 05/15/2024).

[Gui20] Guild of Weavers. *genSTARK*. GitHub. 2020. URL: https://github.com/GuildOfWeavers/genSTARK.

[GWC19] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. *PLONK: Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge*. Cryptology ePrint Archive, Paper 2019/953. 2019. URL: https://eprint.iacr.org/2019/953.

[Kos+20] Ahmed Kosba et al. *MIRAGE: Succinct Arguments for Randomized Algorithms with Applications to Universal zk-SNARKs*. Cryptology ePrint Archive, Paper 2020/278. 2020. URL: https://eprint.iacr.org/2020/278.

[KPS18] Ahmed Kosba, Charalampos Papamanthou, and Elaine Shi. "xJsnark: A Framework for Efficient Verifiable Computation". In: *2018 IEEE Symposium on Security and Privacy (SP)*. 2018, pp. 944–961. DOI: 10.1109/SP.2018.00018.

[KST21] Abhiram Kothapalli, Srinath Setty, and Ioanna Tzialla. *Nova: Recursive Zero-Knowledge Arguments from Folding Schemes*. Cryptology ePrint Archive, Paper 2021/370. 2021. URL: https://eprint.iacr.org/2021/370.

[Kyb24] Kyberturvallisuuskeskus. *Electronic identification*. 2024. URL: https://www.kyberturvallisuuskeskus.fi/en/our-activities/regulation-and-supervision/electronic-identification.

[Lab19] Matter Labs. *Hodor*. GitHub. 2019. URL: https://github.com/matter-labs/hodor.

[Lab23a] Lurk Lab. *Lurk Circuit Specification*. Tech. rep. Lurk Lab, 2023. URL: https://blog.lurk-lang.org/posts/circuit-spec/.

[Lab23b] O(1) Labs. *o1js Documentation*. Tech. rep. O(1) Labs, 2023. URL: https://www.npmjs.com/package/o1js.

[Lab24] O(1) Labs. *Mina Protocol*. 2024. URL: https://minaprotocol.com/.

[Mal+19] Mary Maller et al. "Sonic: Zero-Knowledge SNARKs from Linear-Size Universal and Updatable Structured Reference Strings". In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. CCS '19. London, United Kingdom: Association for Computing Machinery, 2019, pp. 2111–2128. ISBN: 9781450367479. DOI: 10.1145/3319535.3339817. URL: https://doi.org/10.1145/3319535.3339817.

[Min22] Tran Anh Minh. *Theoretical and practical introduction to ZK-SNARKs and ZK-STARKs*. 2022.

[MSV19] Juha Mitrunen, Timo Salovaara, and Janne Viskari. *Electronic identification Report on current status and needs for development*. 2019. URL: https://julkaisut.valtioneuvosto.fi/bitstream/handle/10024/161432/20_2019_Sahkoinen%5C%20tunnistaminen.pdf?sequence=1&isAllowed=y.

[MT21] Dimitris Mouris and Nektarios Georgios Tsoutsos. "Zilch: A Framework for Deploying Transparent Zero-Knowledge Proofs". In: *IEEE Transactions on Information Forensics and Security* 16 (2021), pp. 3269–3284. DOI: 10.1109/TIFS.2021.3074869.

[Nes22] Härmel Nestra. *Zero-Knowledge Proofs for mDL Authentication*. Tech. rep. Cybernetica, 2022. URL: https://cyber.ee/uploads/Zero_Knowledge_Proofs_report_89d6bc5438.pdf.

[Ope] OpenWallet Foundation Labs. *Identity Credential*. URL: https://github.com/openwallet-foundation-labs/identity-credential (visited on 05/15/2024).

[Ope23] Open letter. *Joint statement of scientists and NGOs on the EU's proposed eIDAS reform*. 2023. URL: https://nce.mpi-sp.org/index.php/s/cG88cptFdaDNyRr.

[Par+16] Bryan Parno et al. “Pinocchio: nearly practical verifiable computation”. In: *Commun. ACM* 59.2 (Jan. 2016), pp. 103–112. ISSN: 0001-0782. DOI: 10.1145/2856449. URL: https://doi.org/10.1145/2856449.

[PL] Marika Pietiläinen and Petra Lautamäki. *DVV:n esimerkkisovellus / Finnish EUDI Wallet Demo mobile application*. URL: https://wiki.dvv.fi/pages/viewpage.action?pageId=235522018 (visited on 05/15/2024).

[Reb23] Raul-Martin Rebane. *ZK-SecreC: a Domain-Specific Language for Zero-Knowledge Proofs*. 2023. URL: https://www.youtube.com/watch?v=xnpqbMBd3ig.

[Set20] Srinath Setty. “Spartan: Efficient and General-Purpose zkSNARKs Without Trusted Setup”. In: *Advances in Cryptology – CRYPTO 2020*. Ed. by Daniele Micciancio and Thomas Ristenpart. Cham: Springer International Publishing, 2020, pp. 704–737. ISBN: 978-3-030-56877-1.

[Sig21] Signicat. *The state of digital identity in the Nordics 2021*. 2021. URL: https://f.hubspotusercontent20.net/hubfs/5310879/Digital%5C%20eIDs%5C%20in%5C%20the%5C%20Nordics_14dec2021_v2.pdf.

[SML18] Gordon Stewart, Samuel Merten, and Logan Leland. “Snårkl: Somewhat Practical, Pretty Much Declarative Verifiable Computing in Haskell”. In: Jan. 2018, pp. 36–52. ISBN: 978-3-319-73304-3. DOI: 10.1007/978-3-319-73305-0_3.

[Sta20a] International Organization for Standardization. *Personal identification — ISO-compliant driving licence: Part 5: Mobile driving licence (mDL) application*. ISO/IEC DIS 18013-5. Vernier, Geneva, Switzerland: International Organization for Standardization, 2020. URL: https://www.iso.org/standard/69084.html.

[Sta20b] StarkWare. *Hello, Cairo*. StarkWare. 2020. URL: https://www.cairo-lang.org/hello-cairo/.

[Sta23] StarkWare. *Cairo Documentaion*. Tech. rep. StarkWare, 2023. URL: https://docs.cairo-lang.org/index.html.

[Suo23] Suomi.fi. *General information about your data in registers*. 2023. URL: https://www.suomi.fi/instructions-and-support/your-data/general-information-about-your-data-on-registers.

[Suo24] Suomi.fi. *Suomi.fi e-Authorizations*. 2024. URL: https://liityntakatalogi.suomi.fi/en_GB/dataset/rolesauthsservice.

[SUP14] SUPERFLEX. 2014. URL: https://superflex.net/works/all_data_to_the_people.

[Wah+15] Riad Wahby et al. “Efficient RAM and Control Flow in Verifiable Outsourced Computation”. In: Jan. 2015. DOI: 10.14722/ndss.2015.23097.

[Wah+18] Riad S. Wahby et al. “Doubly-Efficient zkSNARKs Without Trusted Setup”. In: *2018 IEEE Symposium on Security and Privacy (SP)*. 2018, pp. 926–943. DOI: 10.1109/SP.2018.00060.

[Wal] Dan Wallach. *Securing Information for Encrypted Verification and Evaluation (SIEVE)*. URL: https://www.darpa.mil/program/securing-information-for-encrypted-verification-and-evaluation (visited on 05/15/2024).

[Wen+20] Chenkai Weng et al. *Wolverine: Fast, Scalable, and Communication-Efficient Zero-Knowledge Proofs for Boolean and Arithmetic Circuits*. Cryptology ePrint Archive, Paper 2020/925. 2020. URL: https://eprint.iacr.org/2020/925.

[Wen+21] Chenkai Weng et al. *Mystique: Efficient Conversions for Zero-Knowledge Proofs with Applications to Machine Learning*. Cryptology ePrint Archive, Paper 2021/730. 2021. URL: https://eprint.iacr.org/2021/730.

[WMK16] Xiao Wang, Alex J. Malozemoff, and Jonathan Katz. *EMP-toolkit: Efficient MultiParty computation toolkit*. 2016. URL: https://github.com/emp-toolkit.

[Wu23] Howard Wu. *Leo Documentation*. Tech. rep. Aleo, 2023. URL: https://developer.aleo.org/leo/.

[Yan+21] Kang Yang et al. *QuickSilver: Efficient and Affordable Zero-Knowledge Proofs for Circuits and Polynomials over Any Field*. Cryptology ePrint Archive, Paper 2021/076. 2021. DOI: 10.1145/3460120.3484556. URL: https://eprint.iacr.org/2021/076.

[ZCa22] ZCash. *The halo2 Book*. 2022. URL: https://zcash.github.io/halo2/index.html.

[Zha+18] Yupeng Zhang et al. “vRAM: Faster Verifiable RAM with Program-Independent Preprocessing”. In: *2018 IEEE Symposium on Security and Privacy (SP)*. 2018, pp. 908–925. DOI: 10.1109/SP.2018.00013.

[Zha+19] Jiaheng Zhang et al. *Transparent Polynomial Delegation and Its Applications to Zero Knowledge Proof*. Cryptology ePrint Archive, Paper 2019/1482. 2019. URL: https://eprint.iacr.org/2019/1482.

Appendix

I. Licence

Non-exclusive licence to reproduce thesis and make thesis public

I, **Uuna Saarela**,
*(*author's name)

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to

   reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,

   Practical Zero-Knowledge within the European Digital Identity Framework: Implementing Privacy-Preserving Identity Checks,
   *(*title of thesis)

   supervised by Peeter Laud and Chris Brzuska.
   *(*supervisor's name)

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.

4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Uuna Saarela
15/05/2024