



decode

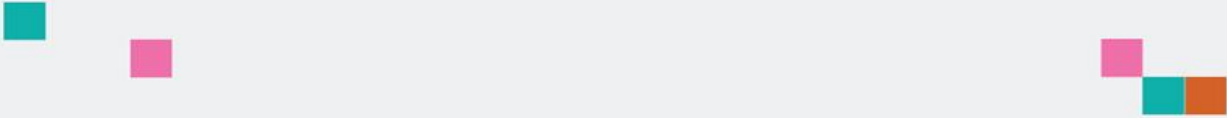


D4.12

Smart Contracts for Data
Commons integrated with
GDPR compliant legal
rules and tested in
pilots



This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no. 732546





Project no. 732546

DECODE

DEcentralised Citizens Owned Data Ecosystem

D4.12 Smart Contracts for Data Commons integrated with GDPR compliant legal rules and tested in pilots

Version Number: V1.0

Lead beneficiary: DYNE.ORG

Due Date: 30/09/19

Author(s): Andrea D'Intino, Denis Roio (DYNE)

Editors and reviewers: Oleguer Sagarra (DRIBIA), Jim Barrit (TW), Eleonora Bassi (POLITO)

Dissemination level: PU		
PU	Public	X
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Approved by: Francesca Bria (DECODE Project Coordinator)

Date: 15/10/2019

This report is currently awaiting approval from the EC and cannot be not considered to be a final version.

Table of Contents

Introduction	4
Multiplatform environment	5
Zencode memory model	6
The “Given” phase	7
The When phase	8
The Then phase	8
The “Coconut” flow	9
GDPR and security in the pilots	11
DDDC Pilot	11
DDDC Pilot smart contracts	13
IoT/BCNNow Pilot	15
18+ Pilot	17
Gebiedonline Pilot	18
Conclusions	19

Introduction

An important outcome in DECODE is having developed and deployed a reliable virtual machine (Zenroom) that can execute smart-contracts written in a human comprehensible language (Zencode) so that participants (user data providers) can consciously define what use is made of their data by services (user data recipients) aggregating and processing them.

We have developed the Zenroom virtual machine and the Zencode language to securely process signed credentials and blind proofs to comply not only with blockchain architectures, but also with privacy regulations as GDPR and in general following privacy by design principles (D1.3).

Here we make a brief overview of the tools and then demonstrate how they apply to specific pilot cases with an analysis of the privacy implications that references the GDPR and in particular findings of DECODE's deliverables (D1.6 and D1.8) and reference to pilots in Amsterdam (D5.5) and Barcelona (D5.6).

The technical delivery of this demonstration consists in the release of Zenroom 1.0.0 and its full documentation attainable at dev.zenroom.org along with the infrastructure running for the pilots and the public workshops in which we provide a public demonstration of this setup.

Multipatform environment

All the smart contracts used in DECODE are executed by Zenroom: its high portability has played a pivotal role in the development of the three pilots using it.

Each of the three pilots using Zenroom (DDDC, IoT/BCNNow, 18+) required software written for different platforms (Linux AMD64, Windows AMD64, MacOS AMD64, Android ARMHF, iOS ARMHF) in different programming languages (Python, Go, JavaScript). For this purpose, different Zenroom's bindings have been developed and a continuous integration platform has been setup to guarantee automated advanced testing and code building on a daily base, using Dyne.org Jenkins platform¹:

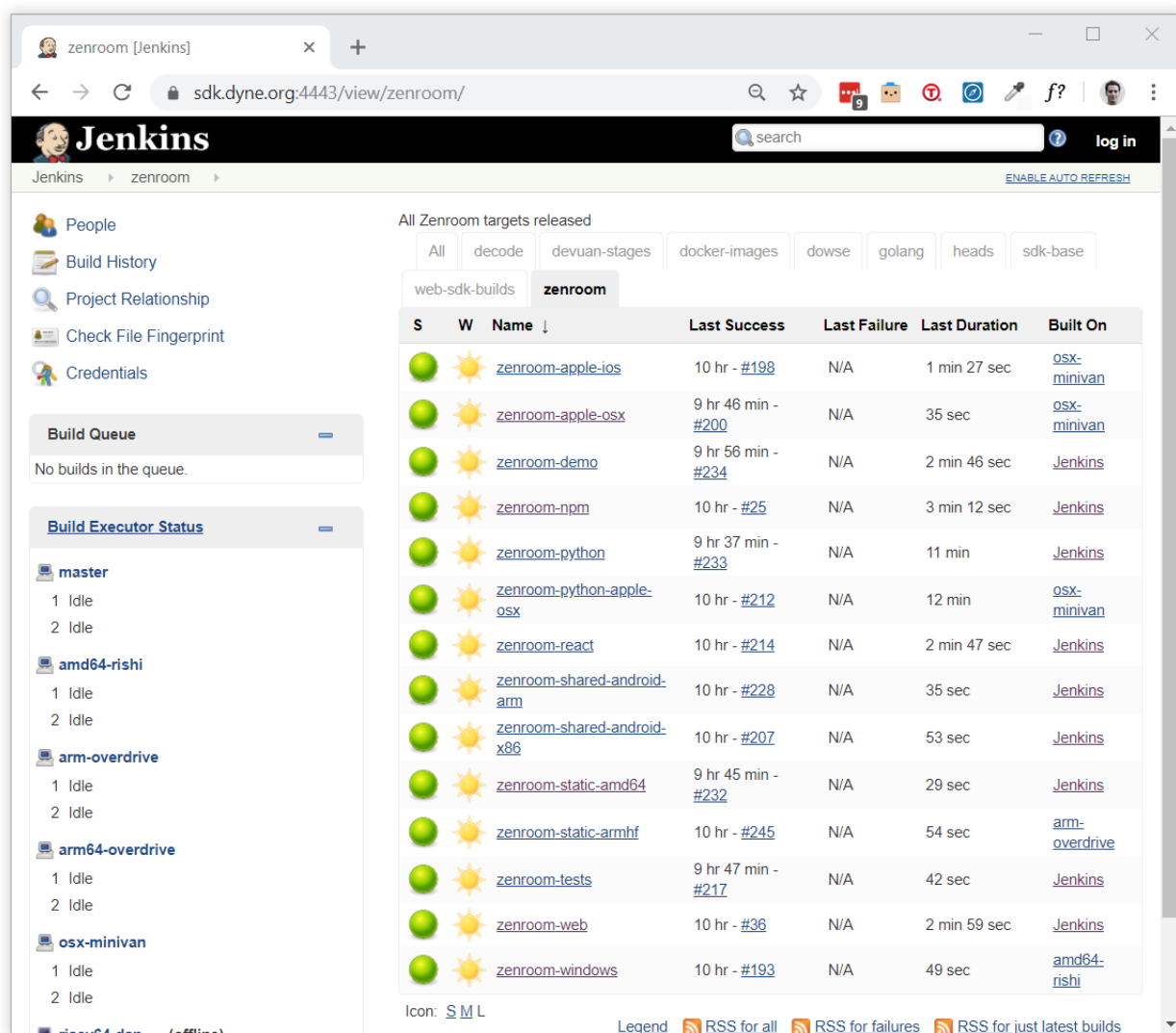


Figure 1: Dyne.org's CI platform

¹ Dyne.org CI: <https://sdk.dyne.org:4443>

Zencode memory model

The Zenroom cryptographic virtual machine includes the interpreter of “Zencode”, a smart-contracts programming language inspired by the “Behavioral Driven Development” (BDD)² and the LANGSEC³ paradigms, aimed to be understandable to English speaker with no software development skills⁴. The Zencode domain specific language has been developed to enable lawyers, business men or non-programmers in general, to write or understand smart contracts, at the same time to allow developers to get closer to end users and provide an effective way to accelerate development.

Zenroom processes Zencode as a finite state machine, operating in three phases, to each correspond three separate blocks of code and three separate memory areas, sealed by security measures⁵. If any single line in a Zencode contract fails, Zenroom stops executing and returns the error.

The first line of a smart-contract written in Zencode must include the version of Zenroom needed to process it, the second line must include the definition of the “Scenario”, which indicates what set of commands will be used, examples of valid scenarios are “simple” and “coconut”. Then the core of the smart-contract ensues, where the three phases are introduced by the prefixes “Given”, “When” and “Then”, followed by commands and variables to compose statements such as:

```
Rule check version 1.0.0
```

```
Scenario simple: Decrypt the message with the password
```

```
Given I have a valid 'secret message'
```

```
When I write 'my secret word' in 'password'
```

```
and I decrypt the secret message with 'password'
```

```
Then print as 'string' the 'text' inside 'message'
```

² BDD introduction, (North): <https://web.archive.org/web/20150901151029/http://behaviourdriven.org/>

³ LANGSEC introduction: <http://langsec.org/>

⁴ Smart contracts for the English speaker (DECODE Blog): <https://decodeproject.eu/blog/smart-contracts-english-speaker>

⁵ Zencode documentation: <https://dev.zenroom.org/zencode/>

and print as 'string' the 'header' inside 'message'

The three phases operate as following:

The “Given” phase

In this phase the input is read from the input, loaded in memory, processed and validated. Zenroom can take input from the stdin⁶ or it accepts two files as input using the interchangeable parameters “-a” and “-k”.

After parsing, if the input contains structured data, the data structure is compared to the object expected based on the definition of the command in the line, embedded in the Zencode parser. If the data structure of the input matches the data structure expected, the data content can be further validated if further conditions are implemented. For example, in the fourth line of smart-contract:

```
Rule check version 1.0.0
```

```
Scenario 'simple': Alice signs a message for Bob
```

```
Given that I am known as 'Alice'
```

```
and I have my valid 'keypair'
```

```
When I write 'This is my signed message to Bob.' in 'draft'
```

```
and I create the signature of 'draft'
```

```
Then print my 'signature'
```

```
and print my 'draft'
```

Zenroom expects to receive a “valid ‘keypair’” tagged with “Alice” as input, therefore it will try to match all the input with the data structure:

⁶ Stdin, [https://en.wikipedia.org/wiki/Standard_streams#Standard_input_\(stdin\)](https://en.wikipedia.org/wiki/Standard_streams#Standard_input_(stdin))

Wikipedia:

```
{ "Alice":  
  
  { "keypair":  
  
    { "private_key": "u64:Yv19MT-yQFC1rm5sRLw6PW619BV6DPKRYmolKCn_Q3k",  
  
      "public_key":  
        "u64:BC0eyWOzCbBHvgkA0ZkgXeRFRwr5N2j2h4Wze48yjg_tKqRGcXT37ppC5CTrb4mJk1O5Q  
5x54-YjSP-xMK_LtLY5eGDcnVvPD-COTiz47PUAiGtuOUUsS0OzxSGft0aJ-A" } } }
```

Any error in the parsing, processing and matching of the input will stop the computation and result in teardown of Zenroom. The “Given” phase does not process or execute commands.

The When phase

In the “When” phase the Zenroom virtual machine moves the data imported in the “Given” phase to its own, separated memory block. This phase allows to process and execute commands: since Zenroom’s memory model makes it completely isolated from the OS, the “When” phase cannot load any input from anywhere else than the “Given” phase, meaning that it will exclusively be able to manipulate the data loaded from the “Given” phase (e.g.: it will not be able to generate a random number, if a random seed has not been loaded in the “Given” phase).

In the “When” phase, Zencode commands can be passed in an arbitrary sequence, allowing for computationally complex manipulation. The list of valid Zencode commands can be extended by modifying existing scenarios or creating new ones, using Lua⁷ scripts to define the interpretation of commands, where along with the standard Lua commands the developer can use a set of cryptographic functions which rely on the cryptographic primitives contained in the Milagro⁸ library.

The Then phase

In the “Then” phase, the output of the “When” phase is moved to a new memory block, where the whole data (or fragments can) be entirely (or selectively) printed in multiple steps, allowing for a formatting of

⁷ The Lua scripting language: <https://www.lua.org/>

⁸ Apache Milagro crypto library: <https://github.com/apache/incubator-milagro-crypto>

the data computed in the “When” phase, which will then be printed to stdout⁹.

The Zencode processing memory structure looks like

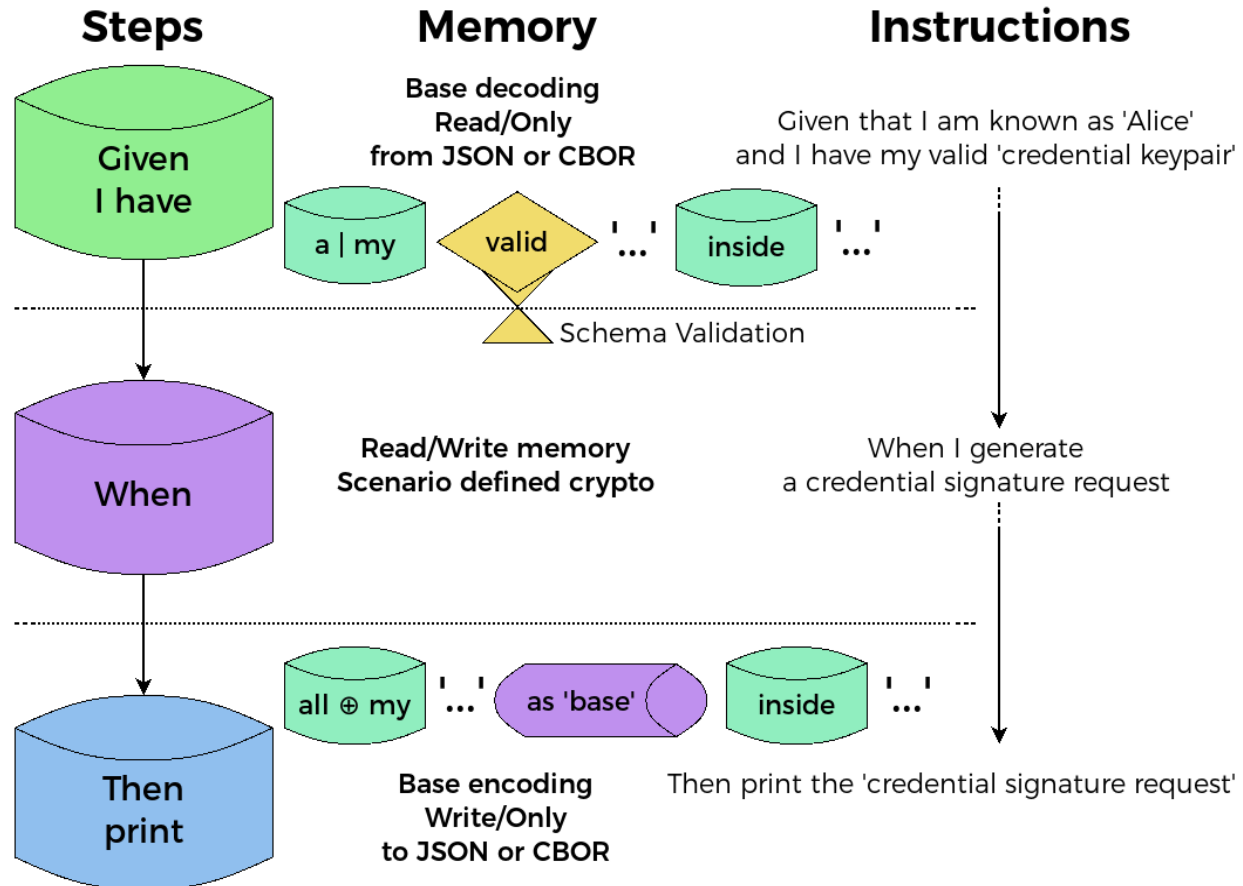


Figure 2: Zencode memory management scheme

The “Coconut” flow

Zenroom implemented the zero-knowledge proof flow described in the “Coconut” paper¹⁰ written by Sonnino et al. at UCL.

⁹ Stdout, [https://en.wikipedia.org/wiki/Standard_streams#Standard_output_\(stdout\)](https://en.wikipedia.org/wiki/Standard_streams#Standard_output_(stdout))

Wikipedia:

¹⁰ “Coconut: Threshold Issuance Selective Disclosure Credentials with Applications to Distributed Ledgers” (Sonnino et al.) : <https://arxiv.org/pdf/1802.07344.pdf>

The Coconut flow allows the “holder” (or “participant” or more generically the “user”), to cryptographically sign objects, using a digital credential, signed by a trusted authority (for example a municipality or more generically someone who is known and trusted by a community), which gets anonymized via randomization at each use (generating a “proof”).

Therefore, using of the Coconut zero-knowledge proof, allows a user to be authenticated even though none of the data, produced and processed in the flow, do contains personal data, thus falling in the “privacy by default” GDPR scenario, which makes the application GDPR compliant without further development.

The Coconut flow can be represented with the scheme:

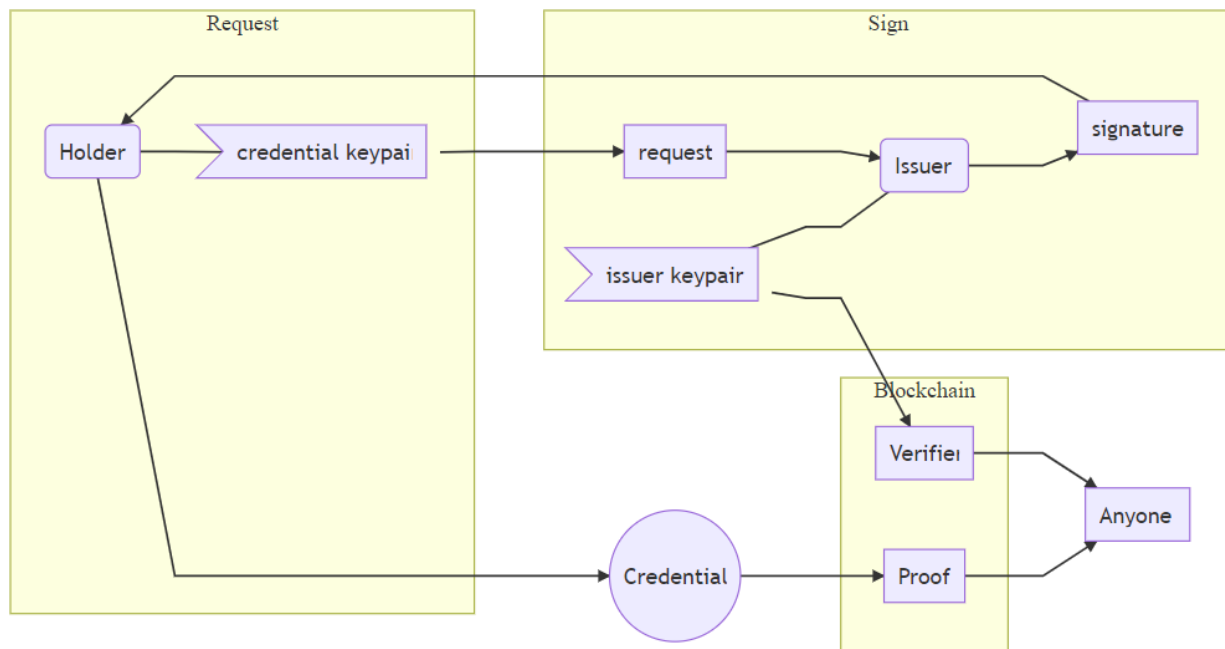


Figure 3: Coconut Zero-knowledge proof flow

GDPR and security in the pilots

Making the usage smart contracts compliant with GPDR depends on several technical factors which in turn differ greatly between different pilots, therefore the analysis will be centered around the pilot's use cases.

DDDC Pilot

The DDDC Pilot allows citizens to digitally sign petitions using an app¹¹, the signature is then stored onto a blockchain. The article 17, par.1 ¹² of GDPR defines the "right to be forgotten" and states:

The data subject shall have the right to obtain from the controller the erasure of personal data concerning him or her without undue delay and the controller shall have the obligation to erase personal data without undue delay [...]

Since blockchains form an immutable historic record¹³, where nor the controller nor anybody will singlehandedly be able to erase data, their usage to store personal data would breach the article 17. For this reason, choses were made in the architecture and the flow to completely avoid the storing of personal data, implementing "privacy by design" and "privacy by default" principles as highlighted in the article 25¹⁴.

The DDDC Pilot relies on the cryptography described in the "Coconut" paper¹⁵ and implemented in Zenroom¹⁶ where they are executed by smart-

¹¹ DECODE app on the Android store: https://www.google.com/url?q=https://play.google.com/store/apps/details?id%3Dcom.dria.decodeapp%26hl%3Den_US&sa=D&ust=1570192213728000&usq=AFQjCNF_VwPdGzoRZx6d2StxHthOxi-pnA

¹² Article 17 of GDPR: <https://eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=CELEX:32016R0679&from=EN#d1e2606-1-1>

¹³ "CONCEPTUALIZING BLOCKCHAINS: CHARACTERISTICS & APPLICATIONS", (Sultan et alt.: par, 2.1) <https://arxiv.org/ftp/arxiv/papers/1806/1806.03693.pdf>

¹⁴ Article 25 of GDPR: <https://eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=CELEX:32016R0679&from=EN#d1e3022-1-1>

¹⁵ "Coconut: Threshold Issuance Selective Disclosure Credentials with Applications to Distributed Ledgers" (Sonnino et al.) : <https://arxiv.org/pdf/1802.07344.pdf>

¹⁶ Source code of the implementation of the Coconut cryptography algorithms in Zenroom: https://github.com/DECODEproject/Zenroom/blob/master/src/lua/zencode_coconut.lua

contracts¹⁷ written in Zencode¹⁸. Then, the security of the processing required by Article 32 of the GDPR is ensured.

More specifically the data flow of the DDDC Pilot generates, stores and processes only:

- A cryptographic keypair for the “citizen”, generated from a random seed by a smart contract executed by an instance of Zenroom running on a mobile device and invoked via a system call¹⁹ by the “DECODE app²⁰”.
- A cryptographic keypair for the “issuer”, generated from a random seed by a smart contract executed by an instance of Zenroom running on a web service invoked via a system call²¹ by the “Credential issuer API²²”.
- The unique id of the petition signed, as it was assigned by the “DDDC Dashboard²³”.
- A “SHA-512²⁴” hash of the petition text, performed by Zenroom.

To summarize, compliance to the articles 5, 17, 25, and 32 of GDPR is guaranteed by:

- In no point of the whole DDDC pilot flow is any personal information stored neither on a blockchain nor in any database system. Aggregated and anonymized data is stored if the user opts-in to do so.

¹⁷ Complete list of the Zencode written smart contracts that operate the DDDC flow, each .zen file is a smart contract: https://github.com/DECODEproject/Zenroom/tree/master/test/zencode_coconut

¹⁸ Zencode developer’s documentation and how-to’s: <https://dev.zenroom.org/zencode>

¹⁹ A “system call” is a technique for an application to execute (an)other application(s) by running a command parsed by a component of the underlying operative system. More info: http://faculty.salina.k-state.edu/tim/ossq/Introduction/sys_calls.html

²⁰ Source code of the DECODE App “V2”: <https://github.com/DECODEproject/decodev2>

²¹ A “system call” is a technique for an application to execute (an)other application(s) by running a command parsed by a component of the underlying operative system. More info: http://faculty.salina.k-state.edu/tim/ossq/Introduction/sys_calls.html

²² Source code of the Credential issuer API: <https://github.com/DECODEproject/credential-issuer>

²³ Source code of the DDDC Dashboard: <https://github.com/DECODEproject/DDDC-instance>

²⁴ “On the Secure Hash Algorithm family”, Penard and Werkhoven: https://web.archive.org/web/20160330153520/http://www.staff.science.uu.nl/~werkh108/docs/study/Y5_07_08/infocry/project/Cryp08.pdf

- No other data is stored that can be paired with any of the data stored in the DDDC flow, which can lead to identification of the natural person, providing full anonymization instead of pseudonymization as defined in Article 4, par. 5²⁵ GDPR.
- Reaching beyond the provisions of GDPR, the DDDC flow insure non-traceability of the citizen, should the citizen happen to use the same credential²⁶ in more than one occasion, by allowing the usage of the randomized “Proof” instead of the unique “Credential”, as described in the “ProveCred” and “VerifyCred” algorithms of the “Coconut” paper.

The platform is however built to be extendable and petitions or services that require storing of personal data on a database can be configured.

DDDC Pilot smart contracts

The smart contracts used in the latest version of DDDC pilot have been integrated in the Zenroom github repository²⁷

Examples of the smart contracts are:

1) Generation of citizen’s keypair²⁸:

```
Scenario coconut: credential keygen
Given that I am known as 'Participant'
When I create the credential keypair
Then print my 'credential keypair'
```

2) Verify the anonymized “Proof” of the citizen’s credential²⁹:

²⁵ Article 4 of GDPR: <https://eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=CELEX:32016R0679&from=EN#d1e1489-1-1>

²⁶ A credential is a cryptographical object issued by the “Credential issuer”, that enables the citizen to be authenticated and is generated by the IssueCred algorithm described in the “Coconut” paper. In the credential both the public ECDH keys of the credential issuer and the citizen are readable.

²⁷ “Zencode Coconut” in the Zenroom repository on github: https://github.com/DECODEproject/Zenroom/tree/master/test/zencode_coconut

²⁸ The smart contract in the Zenroom repository on github: https://github.com/DECODEproject/Zenroom/blob/master/test/zencode_coconut/credential_keygen.zen

Scenario coconut: verify proof

Given that I have a valid 'verifier' from 'Issuer'
and I have a valid 'credential proof'

When I aggregate the verifiers
and I verify the credential proof

Then print 'Success' 'OK' as 'string'

3) Digital signature of the petition³⁰:

Scenario coconut: sign petition

Given I am 'Participant'
and I have my valid 'credential keypair'
and I have a valid 'credentials'
and I have a valid 'verifier' from 'Issuer'

When I aggregate the verifiers
and I create the petition signature 'poll'

Then print the 'petition signature'

²⁹ The smart contract in the Zenroom repository on github:
https://github.com/DECODEproject/Zenroom/blob/master/test/zencode_coconut/verify_proof.zen

³⁰ The smart contract in the Zenroom repository on github:
https://github.com/DECODEproject/Zenroom/blob/master/test/zencode_coconut/sign_petition.zen

IoT/BCNNow Pilot

In the IoT pilot, multifunctional IoT sensors have been distributed to citizens in Barcelona. Zenroom’s smart contracts are used here to encrypt the data, which is later anonymized, implementing “privacy by design” and “privacy by default” principles in line with article 25 of GDPR, making it GDPR compliant.

Zenroom’s smart contracts used in the IoT pilot³¹ provide:

- Generation of keypair for every IoT pilot participant and Point-to-point encryption of the data using the BCNNow public key using Zenroom’s go bindings³² on a web service³³ written in go.
- Managing the “participant” side of the Coconut zero-knowledge proof flow, in order to allow the IoT participant to anonymously get authenticated on the BCNNow website, using Zenroom’s Webassembly port³⁴. The same code is used in the DDDC pilot as well.
- Generation of keypair as well as for the BCNNow server³⁵ and decryption of the IoT data, using Zenroom’s python³⁶ bindings.
- Managing the “credential issuer” side of Coconut zero-knowledge proof flow, in order to allow the IoT participant to anonymously get authenticated on the BCNNow website³⁷, using Zenroom’s python bindings.

³¹ Privacy in the IoT Pilot (contained in “IoT privacy enhancing data sharing: integration with Pilot Infrastructures”, pag.7) <https://decodeproject.eu/publications/iot-privacy-enhancing-data-sharing-integration-pilot-infrastructures>

³² Zenroom go bindings: <https://github.com/DECODEproject/zenroom-go>

³³ IoT encoder source code: <https://github.com/DECODEproject/iotencoder/blob/master/pkg/lua/scripts/encrypt.lua>

³⁴ IoT pilot authentication to BCNNow: <https://github.com/thingful/decodeweb/blob/master/app/assets/js/zenroom/index.js>

³⁵ BCNNow IoT collector: <https://github.com/DECODEproject/bcnnow/blob/master/apps/backend/data/collectors/pull/loTCollector/loTCollector.py>

³⁶ Zenroom’s Python bindings: <https://github.com/DECODEproject/zenroom-py>

³⁷ BCNNow Coconut flow: <https://github.com/DECODEproject/bcnnow/blob/master/apps/backend/data/collectors/pull/DecidimCollector/DecidimPetitionCollector.py>

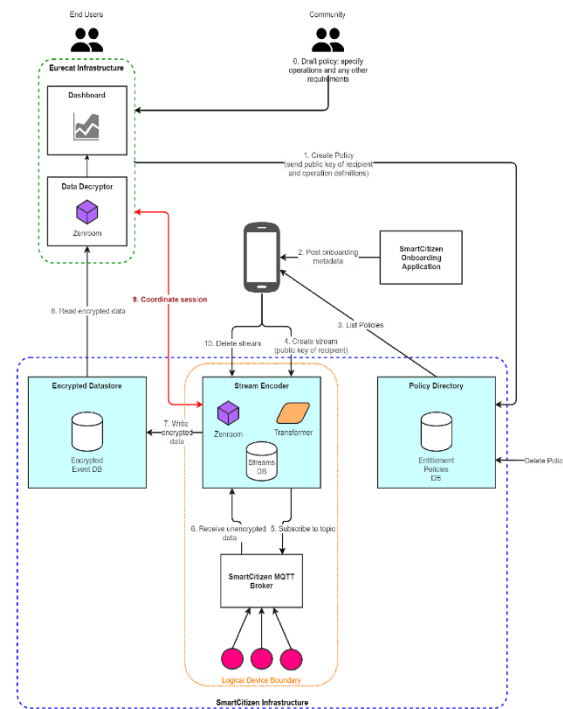
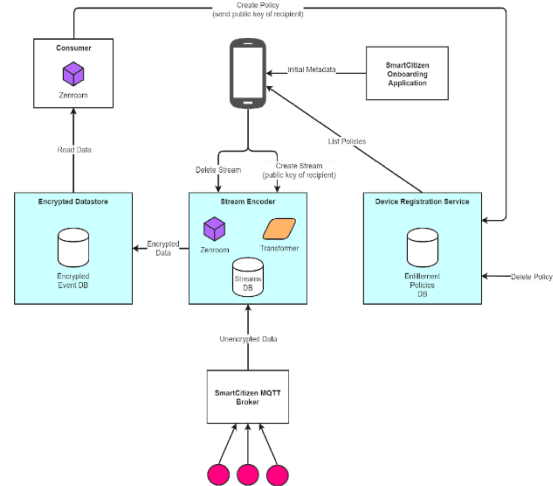


Figure 4: the DECODE IoT Pilot Architecture

In the IoT pilot, Zenroom smart-contracts are responsible for the whole cryptographic flow and authentication, allowing a state-of-the-art level of security. In addition to GDPR compliance, the “The Digital Data Commons Privacy pledge³⁸” has been adopted and is shown on the BCNNow website to the visitors of the website.

³⁸ “Digital Data Commons Privacy Pledge” (contained in the “Licensing of digital commons including personal data – update”, pag. 60): <https://decodeproject.eu/publications/licensing-digital-commons-including-personal-data-update>

18+ Pilot

In Amsterdam's "18+ Pilot" a Zenroom powered passport scanner was developed, to read the NFC chip contained in modern passports. The hardware and software specs are available on the Amsterdam's municipality Github repository³⁹. The scanner looks like:



Figure 5: 18+ scanner box

The scanner produces a QR code, with embedded a credential, that is a JSON object containing:

- The date of birth of the user
- The picture of the user, contained in the passport's rfid chip
- An ECDH signature of the object

The data flow of the 18+ pilot are described in Waag's blog post "Claim Verification 18+: Summary of a DECODE pilot in Amsterdam"⁴⁰ and

³⁹ City of Amsterdam's github repo: https://github.com/Amsterdam/decode_passport_scanner

⁴⁰ "Claim Verification 18+: Summary of a DECODE pilot in Amsterdam" <https://decodeproject.eu/blog/claim-verification-18-summary-decode-pilot-amsterdam>

in the deliverable “Deployment of pilots in Amsterdam” (D5.5)⁴¹. To summarize:

- The QR code is first produced by the screen of the passport scanner
- The QR code is then scanned user’s smartphone, using the web app <https://decode.amsterdam>
- The web app stores the QR in the smartphone’s storage

Since the QR is not processed or stored by any web service, but the processing occurs entirely in the smartphone, the service is fully GDPR compliant.

Gebiedonline Pilot

The Gebiedonline pilot was developed without using Zenroom but using Dutch developed IRMA authentication technology. IRMA offers online, centralized attribute based credential authentication, whose privacy has been audited by an external service provider and is declared on the page <https://wijzinnieuwland.nl/privacy-local>

⁴¹ “Deployment of pilots in Amsterdam” D5.5: <https://www.decodeproject.eu/publications/deployment-pilots-amsterdam>

Conclusions

In three of the four pilots of the project, smart contracts (mostly written in Zencode, some written in Lua) have been extensively used for:

- Implementing an anonymous authentication flow, in line with GDPR “privacy by default” concept (DDDC, IoT/BCNNow, 18+)
- Performing cryptography on data of different nature. (IoT/BCNNow, DDDC)
- Performing the cryptography needed to manage the whole petition flow (creation, signature, tally and count), where each voter’s signature is stored on a blockchain and identified by a randomly generated number, reaching once again GDPR’s “privacy by default”. (DDDC)

The work on licensing for data commons has been included in the BCNNow landing page, that the users access to see how their data is being managed.