DECODE
DEcentralized Citizen Owned Data Ecosystem

Dr. Denis "Jaromil" Roio
Dyne.org CTO & co-founder

Digital Commons & the Future of Cities
NEXA 10th Conference
Polito, Future Urban Legacy Lab
18 December 2018

Hacker community since 1994 – GNU/Linux/BSD

Internet based not-for-profit software foundry

Sustainable tech / Interdisciplinarity / Art & Science

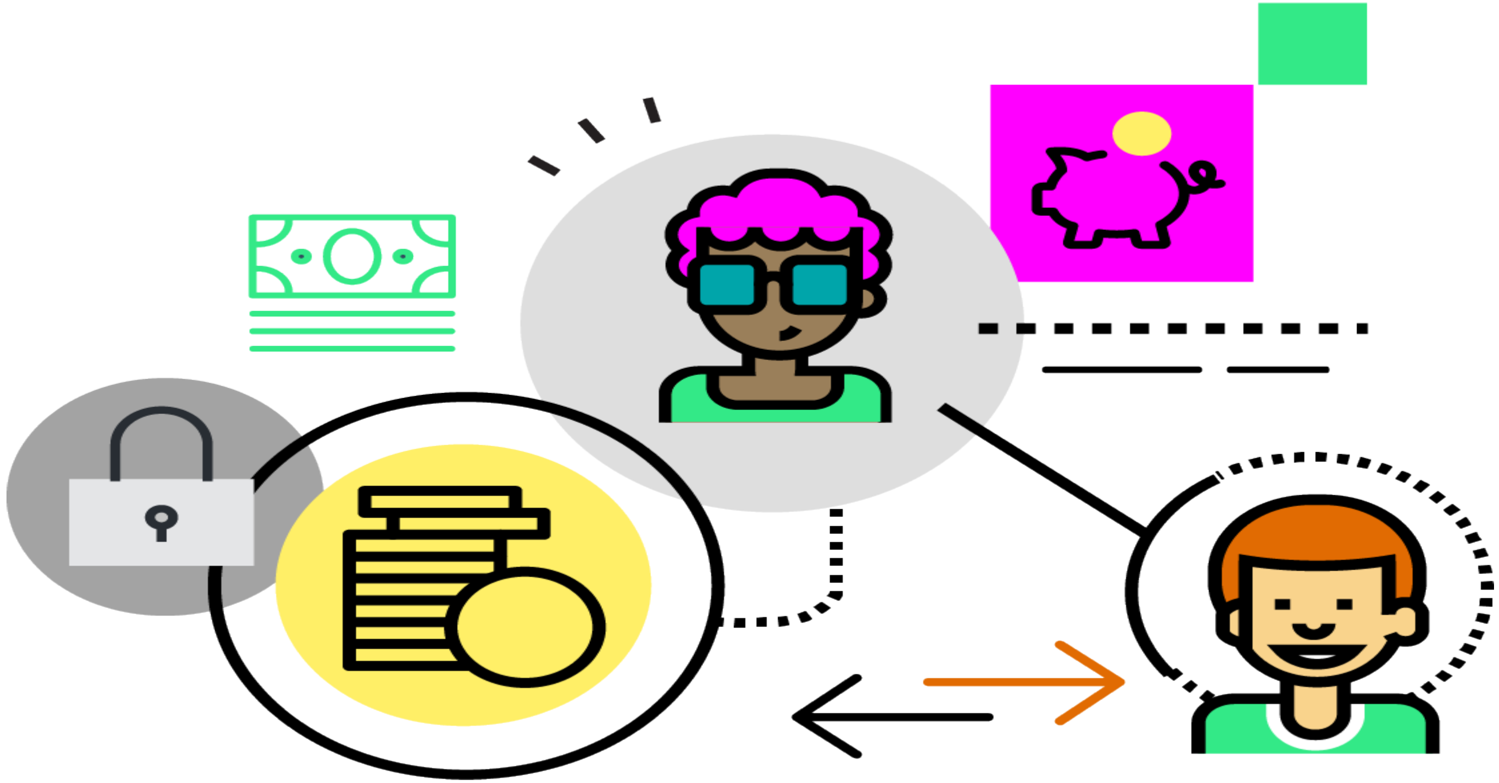
Design with minimalism: UNIX principles

Community engagement

and empowerment

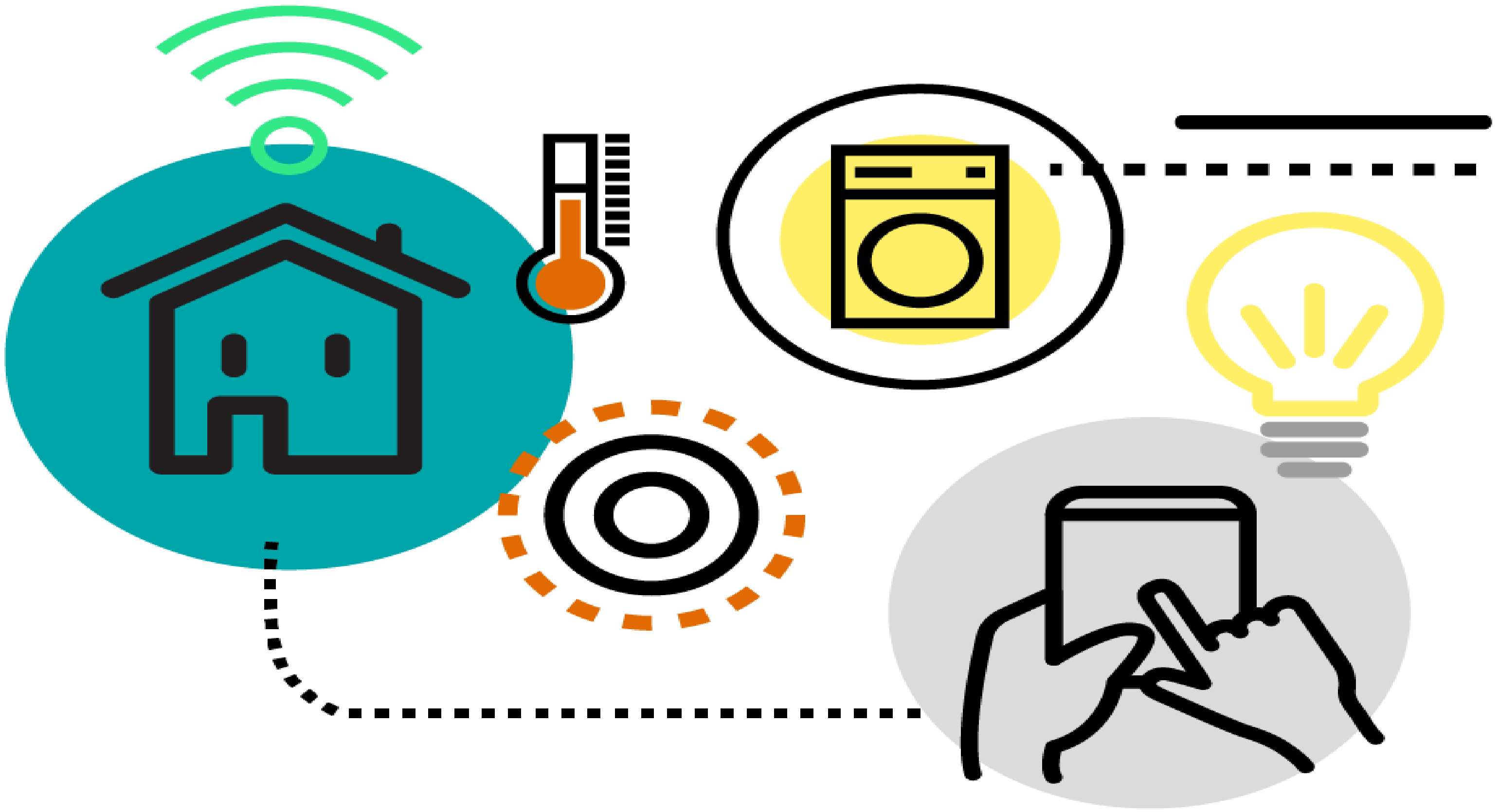
decode

Sharing Economies... ?! but with whom are we really sharing our needs and desires?



decode

Internet of Things... communicating on our behalf... but who are they speaking with and about what?



Distributed Ledger Technology (Blockchain)

Peer 2 Peer

Consensus

decode

Virtual Machine

Ledger

decode

GNU+Linux Operating System
for distributed logical computing,
controlled execution environment.

Minimalist, resource optimised, fully documented, customisable and available to run on cloud, bare metal and more than 30 ARM devices (open hardware!)

...based on:

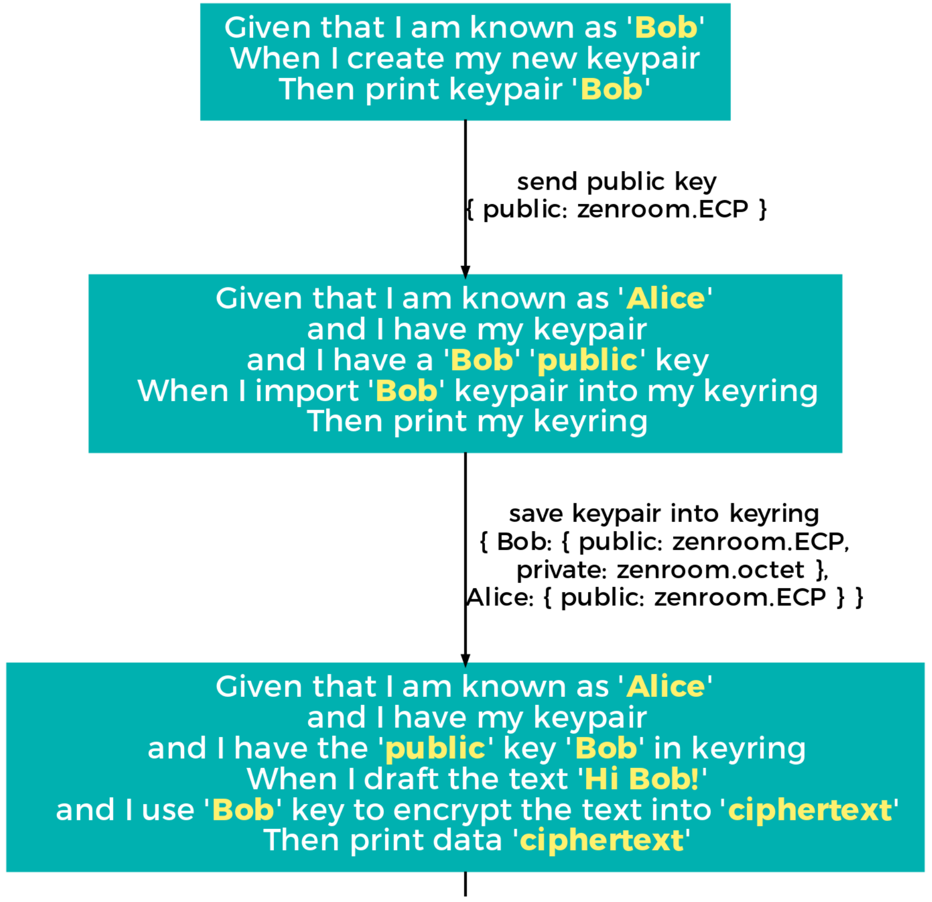
DEVUAN

Smart-rules language:

→ Zenroom Virtual Machine (VM)
→ Zencode Domain Specific Language (DSL)

decode

- Controlled execution and DSL for Elliptic Curve cryptography
- Extremely portable component for end-to-end encryption
- Language theoretic security design co-evolving with pilots
- Facilitates interdisciplinary code reviews



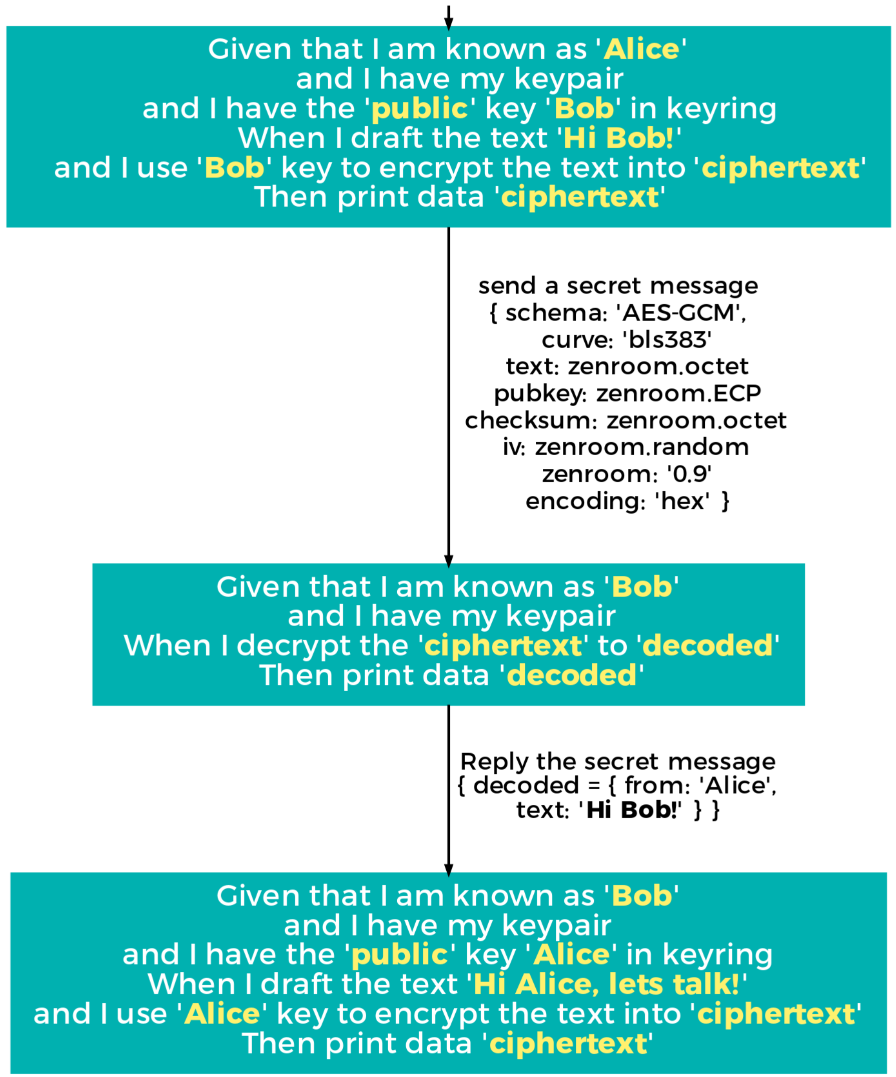
Given that I am known as '**Bob**'
When I create my new keypair
Then print keypair '**Bob**'

send public key
{ public: zenroom.ECP }

Given that I am known as '**Alice**'
and I have my keypair
and I have a '**Bob**' '**public**' key
When I import '**Bob**' keypair into my keyring
Then print my keyring

save keypair into keyring
{ Bob: { public: zenroom.ECP,
private: zenroom.octet },
Alice: { public: zenroom.ECP } }

Given that I am known as '**Alice**'
and I have my keypair
and I have the '**public**' key '**Bob**' in keyring
When I draft the text '**Hi Bob!**'
and I use '**Bob**' key to encrypt the text into '**ciphertext**'
Then print data '**ciphertext**'



Given that I am known as '**Alice**'
and I have my keypair
and I have the '**public**' key '**Bob**' in keyring
When I draft the text '**Hi Bob!**'
and I use '**Bob**' key to encrypt the text into '**ciphertext**'
Then print data '**ciphertext**'

send a secret message
{ schema: 'AES-GCM',
curve: 'bls383'
text: zenroom.octet
pubkey: zenroom.ECP
checksum: zenroom.octet
iv: zenroom.random
zenroom: '0.9'
encoding: 'hex' }

Given that I am known as '**Bob**'
and I have my keypair
When I decrypt the '**ciphertext**' to '**decoded**'
Then print data '**decoded**'

Reply the secret message
{ decoded = { from: 'Alice',
text: '**Hi Bob!**' } }

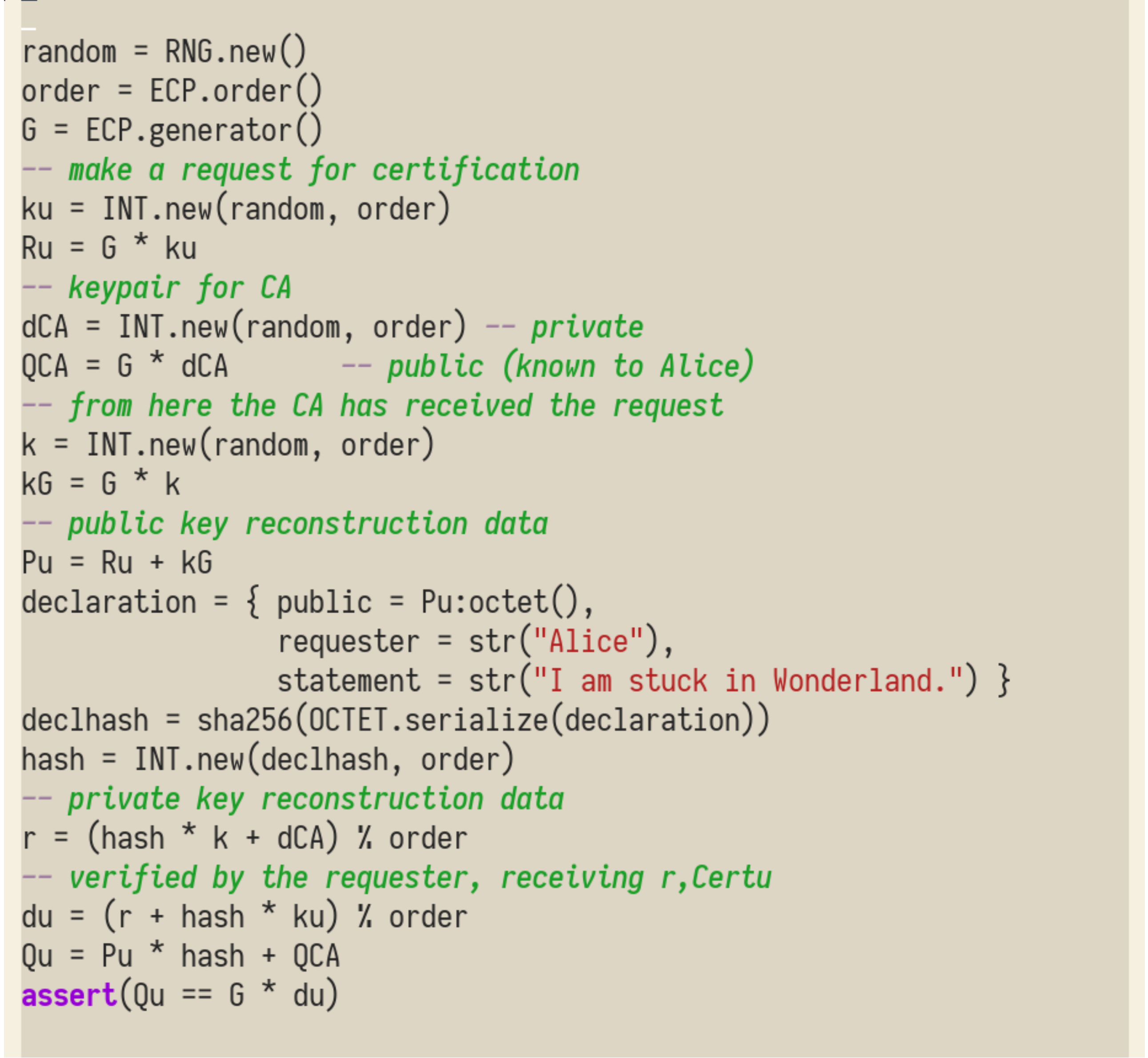
Given that I am known as '**Bob**'
and I have my keypair
and I have the '**public**' key '**Alice**' in keyring
When I draft the text '**Hi Alice, lets talk!**'
and I use '**Alice**' key to encrypt the text into '**ciphertext**'
Then print data '**ciphertext**'

decode

Elliptic Curve Qu-Vanstone Implicit Certificate

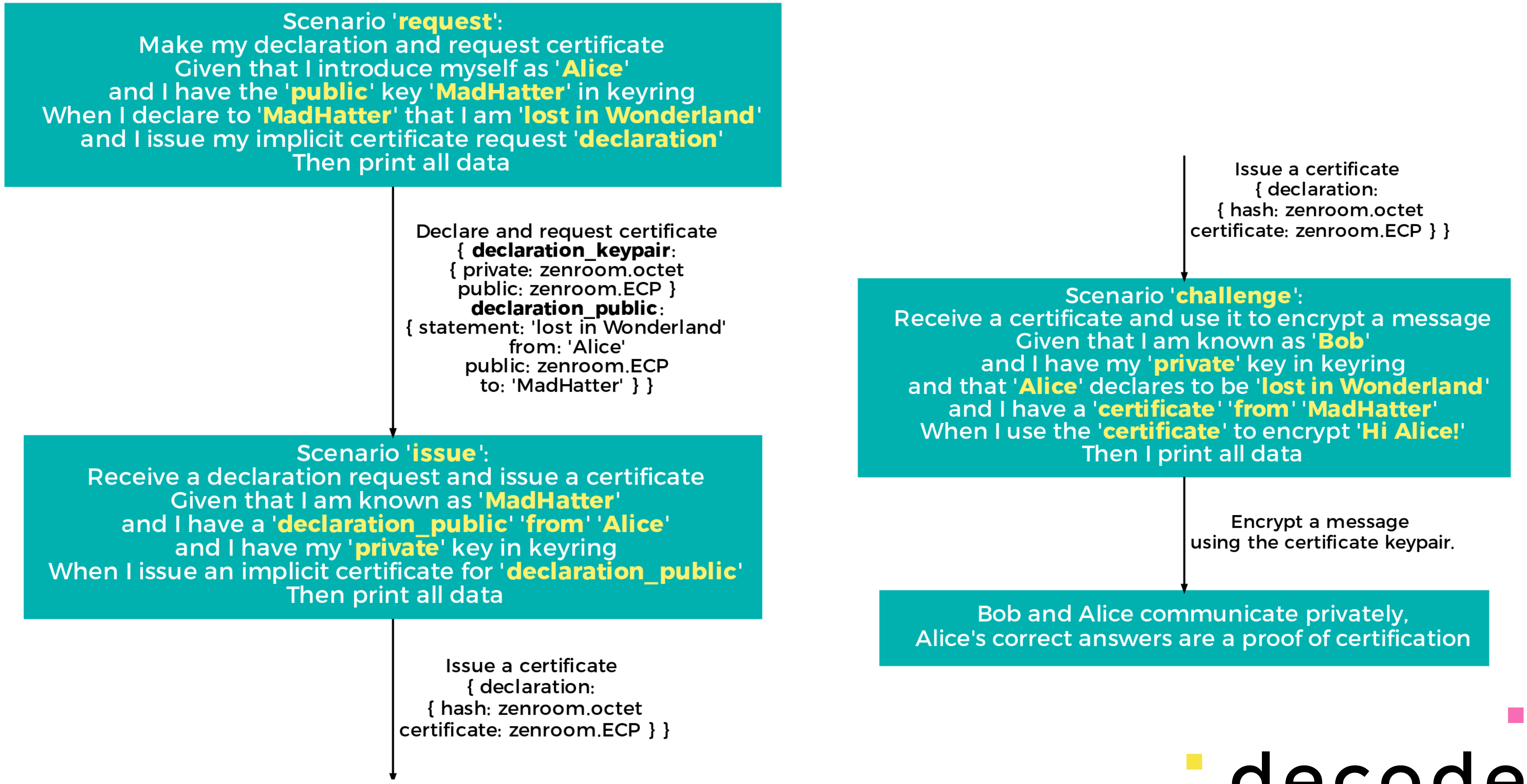
$$
\begin{array}{lcl}
\underline{\qquad\qquad U \qquad\qquad} & & \underline{\qquad\qquad CA \qquad\qquad} \\
k_U \in_R [1, \ldots, n-1] & & \\
R_U := k_U G & & \\
 & \xrightarrow{U,\ R_U} & \\
 & & k \in_R [1, \ldots, n-1] \\
 & & P_U := R_U + kG \\
 & & Cert_U := Encode(P_U, U, *) \\
 & & e := H_n(Cert_U) \\
 & & r := ek + d_{CA} \pmod{n} \\
 & \xleftarrow{r,\ Cert_U} & \\
e := H_n(Cert_U) & & \\
d_U := ek_U + r \pmod{n} & & \\
Q_U := eP_U + Q_{CA} & &
\end{array}
$$

Example of ECQV “implicit certificate” implementation in Zenroom.dyne.org



```
random = RNG.new()
order = ECP.order()
G = ECP.generator()
-- make a request for certification
ku = INT.new(random, order)
Ru = G * ku
-- keypair for CA
dCA = INT.new(random, order) -- private
QCA = G * dCA         -- public (known to Alice)
-- from here the CA has received the request
k = INT.new(random, order)
kG = G * k
-- public key reconstruction data
Pu = Ru + kG
declaration = { public = Pu:octet(),
                requester = str("Alice"),
                statement = str("I am stuck in Wonderland.") }
declhash = sha256(OCTET.serialize(declaration))
hash = INT.new(declhash, order)
-- private key reconstruction data
r = (hash * k + dCA) % order
-- verified by the requester, receiving r,Certu
du = (r + hash * ku) % order
Qu = Pu * hash + QCA
assert(Qu == G * du)
```

decode



Scenario '**request**':
Make my declaration and request certificate
Given that I introduce myself as '**Alice**'
and I have the '**public**' key '**MadHatter**' in keyring
When I declare to '**MadHatter**' that I am '**lost in Wonderland**'
and I issue my implicit certificate request '**declaration**'
Then print all data

Declare and request certificate
{ **declaration_keypair**:
{ private: zenroom.octet
public: zenroom.ECP }
declaration_public:
{ statement: 'lost in Wonderland'
from: 'Alice'
public: zenroom.ECP
to: 'MadHatter' } }

Scenario '**issue**':
Receive a declaration request and issue a certificate
Given that I am known as '**MadHatter**'
and I have a '**declaration_public**' '**from**' '**Alice**'
and I have my '**private**' key in keyring
When I issue an implicit certificate for '**declaration_public**'
Then print all data

Issue a certificate
{ declaration:
{ hash: zenroom.octet
certificate: zenroom.ECP } }

Issue a certificate
{ declaration:
{ hash: zenroom.octet
certificate: zenroom.ECP } }

Scenario '**challenge**':
Receive a certificate and use it to encrypt a message
Given that I am known as '**Bob**'
and I have my '**private**' key in keyring
and that '**Alice**' declares to be '**lost in Wonderland**'
and I have a '**certificate**' '**from**' '**MadHatter**'
When I use the '**certificate**' to encrypt '**Hi Alice!**'
Then I print all data

Encrypt a message
using the certificate keypair.

Bob and Alice communicate privately,
Alice's correct answers are a proof of certification

decode

Look at the future with our expert team

dyne.org

Strategy

Consultancy

Development

Available for workshops, focused meetings and development projects

e-mail: info @ dyne.org