krypt. semper pi.

whoami

ruby-core

ruby openssl

freelancer

crypto is hard

u can't touch this

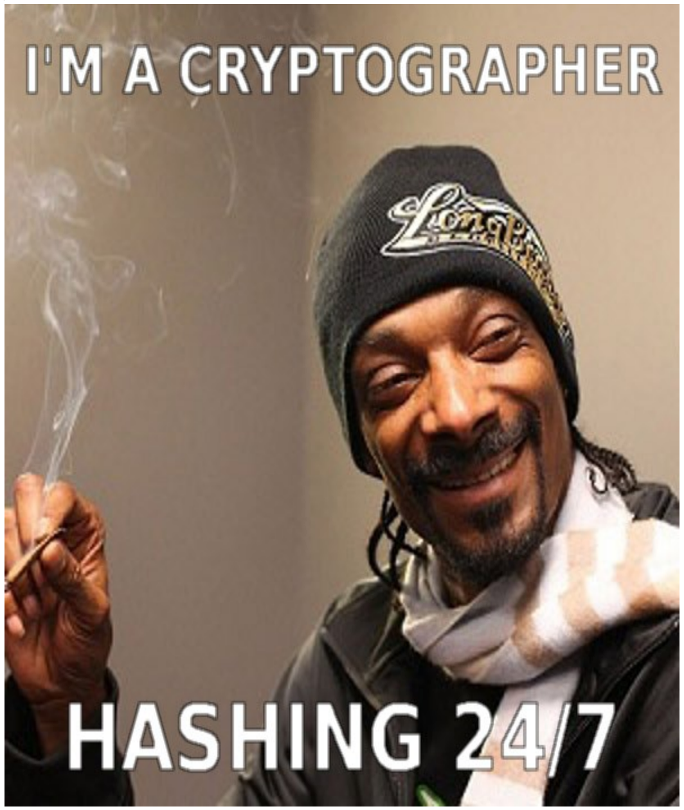
touched crypto

only a select few

can really master

the arcane art that is cryptography

mastered crypto



so, if crypto is hard

do crypto apis have to be, too?

two opposing forces

„when i refactored our login,
i mostly cared about security.
what do i care about algorithms?
i need my time to work out!“

„i did a lot of ssl pentesting lately.
i need full control of all parameters.
and i sure do need my legacy algorithms, dude.“

everyday development:

security sucks

„oh no, not security again!“

security is the jar jar binks
of software development

```
describe Cryptography do

  it 'shouldn't be touched by mere mortals' do
    expect {
      write_crypto(:joe_programmer)
    }.to raise_error(ArgumentError)
  end

  it 'should only be written by experts in the field' do
    expect { hire_expert }.to be_the_default
  end

end
```

```
describe Cryptography do
  context 'reality' do

    let(:expert) { :joe_programmer }

    it 'shouldn't be touched by mere mortals' do
      hope { write_crypto(expert) }.not_to raise_error(ToldYaSo)
     end

     it 'should only be written by experts in the field' do
      expect { hire_expert }.to raise_error(BudgetError)
     end

   end
end
```

so here's your catch-22:

every app needs a security guy

but not
every security guy needs your app

good design

==

making complex things easy

databases are hard -> active record

threads are hard -> stm, actors

celluloid

memory is hard -> gc

why do crypto libraries still

force us to deal with the bare metal?

don't bother me with details

but experts™ need full control

conflict

abstraction vs. oversimplification

there is no golden middle

if full control is your thing:

openssl
java security api
&
friends

if all you care about is security by default:

keyczar (no friends)

(maybe nacl / libsodium)

why not just use keyczar and be done with it?

well, there's the experts™

and legacy apps

we need both

full control if needed

security by default otherwise

„sadly we can't have both. or can we?“

revolutionary idea

combine both aspects in one library

krypt. semper pi.

ok, cool, but...

WTF
IS THIS SHIT?

who we are

@nahi, @vipulnsward, @abstractj, @qmx, @_emboss_

krypt first of all is a framework

using existing libraries

to implement

core cryptography primitives

peaceful coexistence

(at least at first)

expert™

„i see – this is the low-level api that gives me the full control that i so badly need“

digest

cipher

signature

„let's say i start an app from scratch and all i really care about is the thing being secure – what do i do then?“

krypt also offers higher-level protocols

using the lower-level ones as a basis

what's the big deal?

crypto code today

#1 encrypting data

what it should look like

```ruby
require 'openssl'

data = 'le secret'

cipher = OpenSSL::Cipher.new('aes-128-cbc')
cipher.encrypt
key = cipher.random_key
iv = cipher.random_iv

enc = cipher.update(data) + cipher.final


decipher = OpenSSL::Cipher::AES.new('aes-128-cbc')
decipher.decrypt
decipher.key = key
decipher.iv = iv

plain = decipher.update(enc) + decipher.final
```

what it actually looks like

```
require 'openssl'

data = 'le secret'
key = 'lepasswordlepassword'

cipher = OpenSSL::Cipher.new('AES-128-ECB')
cipher.encrypt
cipher.key = key

enc = cipher.update(data) + cipher.final

decipher = OpenSSL::Cipher::AES.new('AES-128-ECB')
decipher.decrypt
decipher.key = key

plain = decipher.update(enc) + cipher.final
```

```
require 'openssl'

data = 'le secret'
key = 'lepasswordlepassword' # fail

cipher = OpenSSL::Cipher.new('AES-128-ECB')
cipher.encrypt
cipher.key = key

enc = cipher.update(data) + cipher.final

decipher = OpenSSL::Cipher::AES.new('AES-128-ECB')
decipher.decrypt
decipher.key = key

plain = decipher.update(enc) + cipher.final
```

```
require 'openssl'

data = 'le secret'
key = 'lepasswordlepassword'

cipher = OpenSSL::Cipher.new('AES-128-ECB') # fail
cipher.encrypt
cipher.key = key

enc = cipher.update(data) + cipher.final

decipher = OpenSSL::Cipher::AES.new('AES-128-ECB') # fail
decipher.decrypt
decipher.key = key

plain = decipher.update(enc) + cipher.final
```

```ruby
require 'openssl'

data = 'le secret'
key = 'lepasswordlepassword'

cipher = OpenSSL::Cipher.new('AES-128-ECB')
cipher.encrypt
cipher.key = key
# no iv -> fail

enc = cipher.update(data) + cipher.final

decipher = OpenSSL::Cipher::AES.new('AES-128-ECB')
decipher.decrypt
decipher.key = key
# no iv -> fail

plain = decipher.update(enc) + cipher.final
```

```ruby
require 'openssl'

data = 'le secret'
key = 'lepasswordlepassword'

cipher = OpenSSL::Cipher.new('AES-128-ECB')
cipher.encrypt # design fail
cipher.key = key

enc = cipher.update(data) + cipher.final

decipher = OpenSSL::Cipher::AES.new('AES-128-ECB')
decipher.decrypt # design fail
decipher.key = key

plain = decipher.update(enc) + cipher.final
```

```
require 'openssl'

data = 'le secret'
key = 'lepasswordlepassword'

cipher = OpenSSL::Cipher.new('AES-128-ECB')
cipher.encrypt
cipher.key = key

enc = cipher.update(data) + cipher.final # design fail

decipher = OpenSSL::Cipher::AES.new('AES-128-ECB')
decipher.decrypt
decipher.key = key

plain = decipher.update(enc) + cipher.final # design fail
```

#2 pbkdf2 password hash

what it should look like

```ruby
require 'openssl'

pass = 'le secret'
salt = OpenSSL::Random.random_bytes(16)
iter = 20000
len = OpenSSL::Digest::SHA1.new.digest_len # 20

hash = OpenSSL::PKCS5.pbkdf2_hmac_sha1(pass, salt, iter, len)
```

what it actually looks like

```
require 'openssl'

pass = 'le secret'
salt = pass
iter = 10
len = password.size

hash = OpenSSL::PKCS5.pbkdf2_hmac_sha1(pass, salt, iter, len)
```

```
require 'openssl'

pass = 'le secret'
salt = pass # fail
iter = 10
len = password.size

hash = OpenSSL::PKCS5.pbkdf2_hmac_sha1(pass, salt, iter, len)
```

```
require 'openssl'

pass = 'le secret'
salt = pass
iter = 10 # fail
len = password.size

hash = OpenSSL::PKCS5.pbkdf2_hmac_sha1(pass, salt, iter, len)
```

```
require 'openssl'

pass = 'le secret'
salt = pass
iter = 10
len = password.size # fail

hash = OpenSSL::PKCS5.pbkdf2_hmac_sha1(pass, salt, iter, len)
```

```
require 'openssl'

pass = 'le secret'
salt = pass
iter = 10
len = password.size
                                                  # design fail
hash = OpenSSL::PKCS5.pbkdf2_hmac_sha1(pass, salt, iter, len)
```

```
require 'openssl'

pass = 'le secret'
salt = pass
iter = 10
len = password.size
                                    o_O
hash = OpenSSL::PKCS5.pbkdf2_hmac_sha1(pass, salt, iter, len)
```

what it actually **actually** looks like

```
require 'openssl'

pass = 'le secret'
salt = pass
digest = OpenSSL::MD5.new

hash = digest.digest(salt + pass)
```

FEYENOORD
ROTTERDAM

#3 certificate validation

what it should look like

```
require 'openssl'


# who the fuck knows?!
```

almost impossible to do it correctly™

online revocation checks

openssl refuses to have dependencies

relic of bygone times

before

dependency management tools

we're left with half-assed validation

krypto code tomorrow

what it should actually look like

#1 encrypting data

```
require 'krypt'

data = 'le secret'

encrypter = Krypt::Encrypter.new
key = encrypter.generate_key

enc = encrypter.encrypt(data)


decrypter = Krypt::Decrypter.new
decrypter.key = key

plain = decrypter.decrypt(data)
```

#2 password hash

```ruby
require 'krypt'

pass = 'le secret'

hash = Krypt::PasswordHash.hash(pass)

begin
  Krypt::PasswordHash.verify(hash)
rescue Krypt::PasswordHash::Error
  # react
end

#swell
```

#3 certificate validation

krypt.semper pi.

```ruby
require 'krypt'

certificate = Krypt::X509::Certificate.new(bytes)

begin
  certificate.verify
rescue Krypt::X509::VerificationError
  # react
end

#swell
```

don't bother me with details

security by default

ONE DOES NOT SIMPLY
REPLACE OPENSSL LIKE THAT

use

protocols

moar advantages:

moar tests

rspec

FuzzBert

moar docs (non-expert™)

moar ruby

easier to understand & maintain

minimal portion of native code

using whatever library is available

in the background

the rest is plain ruby

„but why should anyone care about your stupid crypto library?!“

write once, run anywhere$^{\text{TM}^2}$

run on all rubies

https

if https doesn't work,

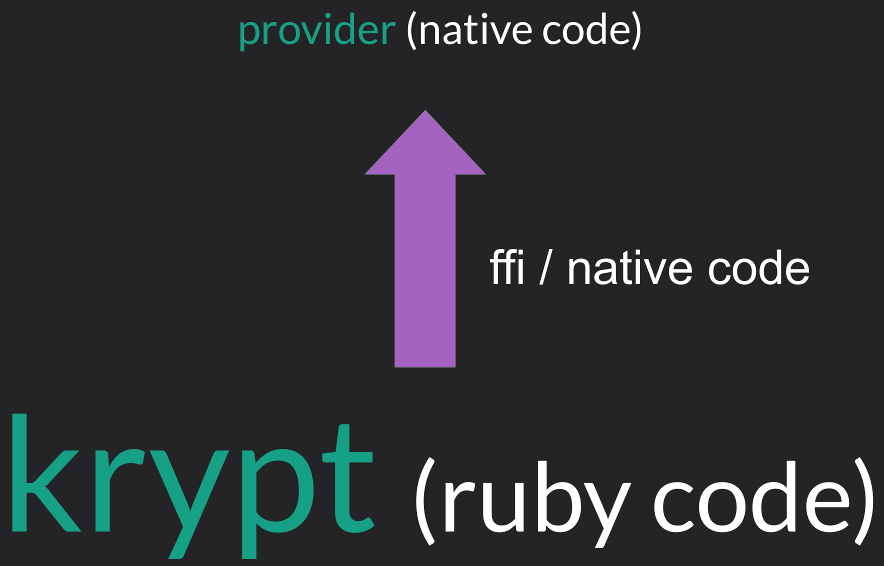
ruby doesn't work

(https://rubygems.org, anyone?)

openssl isn't available everywhere

c (or java) extensions are not the answer only ruby runs everywhere

this is part of java's success


provider (native code)
ffi / native code
krypt (ruby code)

use openssl on jruby

same code runs on all platforms

using different parts to get there

write once, run anywhere$^{\text{TM}^2}$

future: all-ruby provider

„if it runs ruby, it runs krypt“

think:
webrick
vs.
thin, unicorn, puma, torquebox

eat your own dog

...err food

easier said than **done**

„zomg this needs to be fast,

let's write a c extension !!!“

been there, done that

asn.1

c & java implementations

fast

\o/

code duplication

/o\

let's give ruby a try

https://github.com/krypt/krypt-asn1-rb

much less code

that i still understand one week later

MRI

```
Krypt::Asn1.decode String(n=100000)            1.418942
Krypt::Asn1.decode StringIO(n=100000)           1.353945

OpenSSL::ASN1.decode String(n=100000)           4.948085
OpenSSL::X509::Certificate String(n=100000)     3.466104
```

JRUBY

```
Krypt::Asn1.decode String(n=100000)              0.903000
Krypt::Asn1.decode StringIO(n=100000)            0.896000

OpenSSL::ASN1.decode String(n=100000)            7.920000
OpenSSL::X509::Certificate String(n=100000)      8.247000
```

RBX

```
Krypt::Asn1.decode String(n=100000)              2.807615
Krypt::Asn1.decode StringIO(n=100000)            2.535868

OpenSSL::ASN1.decode String(n=100000)           16.184046
OpenSSL::X509::Certificate String(n=100000)     10.918114
```

this is worthy repeating:

we are talking

ruby
vs.
c & java

here

streaming parser
with
lazy evaluation

it's not so much the

programming language,

but the algorithm

that counts

use ruby™

still a long road ahead of us

plan: krypt as default crypto library

moar plan:

use openssl polyfill for the interim period https://github.com/krypt/krypt-ossl

thank you

https://github.com/krypt

http://martinbosslet.de

martin.bosslet@gmail.com

@_emboss_