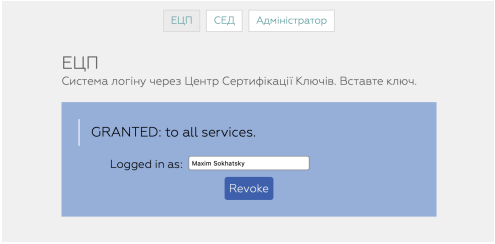
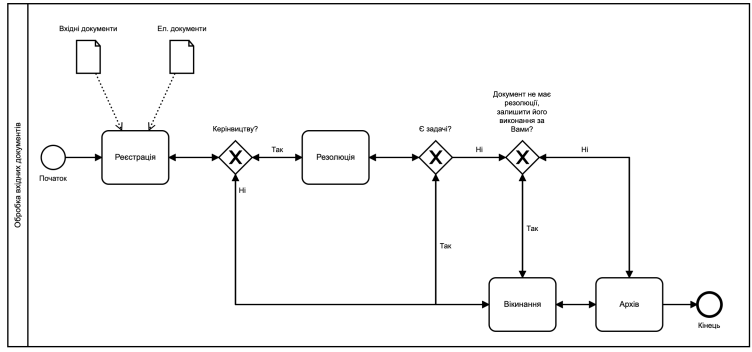
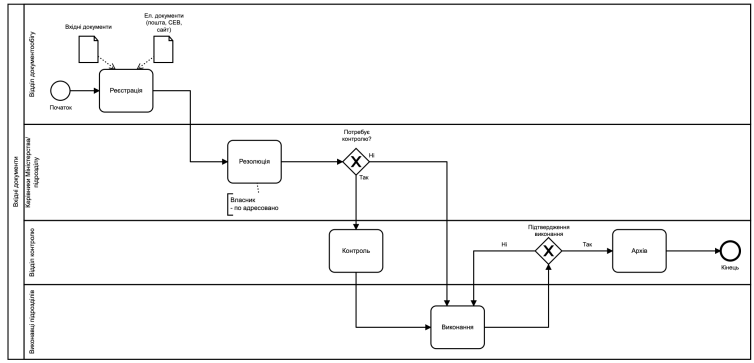
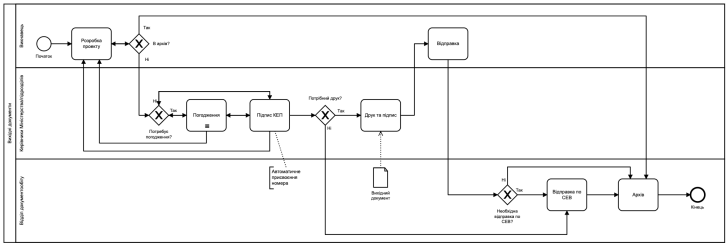
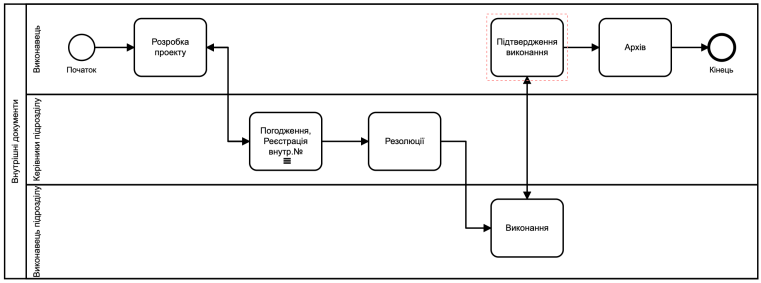
```
def route(<<"ldap", _::binary>>), do: LDAP.Index
def route(<<"crm",  _::binary>>), do: CRM.Index
def route(<<"rmk",  _::binary>>), do: RMK.Index
def route(<<"kvs",  _::binary>>), do: KVS.Index
def route(<<"act",  _::binary>>), do: BPE.Actor
def route(<<"help", _::binary>>), do: HELP.Index
```

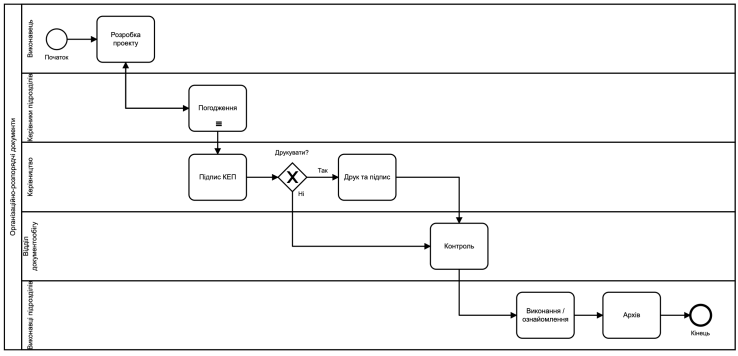


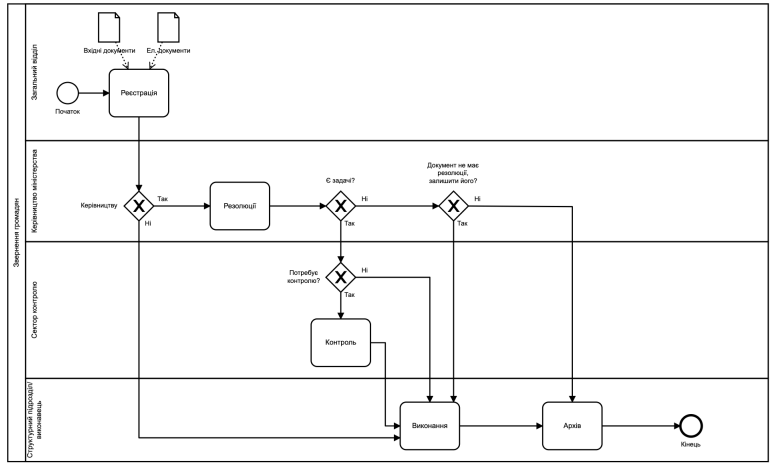

Обробка вхідних документів
Вхідні документи
Ел. документи
Початок
Реєстрація
Керівництву?
Так
Ні
Резолюція
Є задачі?
Ні
Так
Документ не має резолюції, залишити його виконання за Вами?
Ні
Так
Вікинання
Архів
Кінець

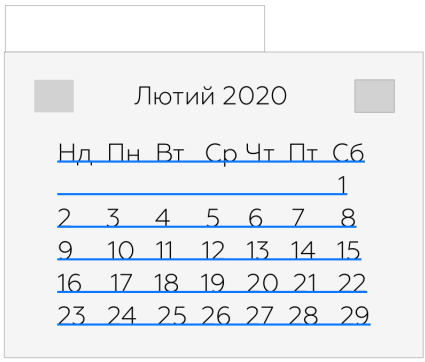

Вхідні документи
Керівництво/Заступники/підрозділу
Відділ контролю
Виконавці / підрозділи
Відділ документообігу
Вхідні документи
Ел. документи (пошта, СЕВ, сайт)
Початок
Реєстрація
Резолюція
Потрибує контролю?
Ні
Так
Власник - по адресовано
Контроль
Підтвердження виконання
Ні
Так
Архів
Кінець
Виконання

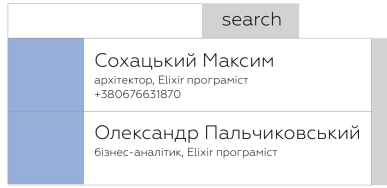

Вихідні документи
Виконавець
Керівник/заступник/керівник
Відділ документообігу
Початок
Розробка проекту
В архів?
Так
Ні
Погодження
Потрібно погодження?
Підпис КЕП
Потрібний друк?
Друк та підпис
Відправка
Автоматичне присвоєння номера
Вихідний документ
Необхідно надіслати по СЕВ?
Відправка по СЕВ
Архів
Кінець

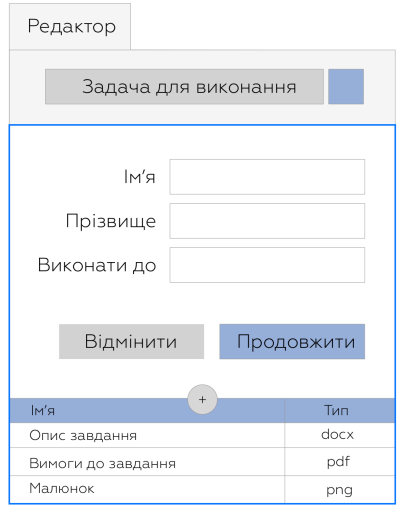

Внутрішні документи
Виконавець
Керівники підрозділу
Виконавець підрозділу
Початок
Розробка проекту
Погодження, Реєстрація внутр.№
Резолюції
Виконання
Підтвердження виконання
Архів
Кінець


Організаційно-розпорядчі документи
Виконавець
Керівник підрозділу
Керівництво
Відділ документообігу
Виконавці підрозділів
Початок
Розробка проекту
Погодження
Підпис КЕП
Друкувати?
Так
Ні
Друк та підпис
Контроль
Виконання / ознайомлення
Архів
Кінець

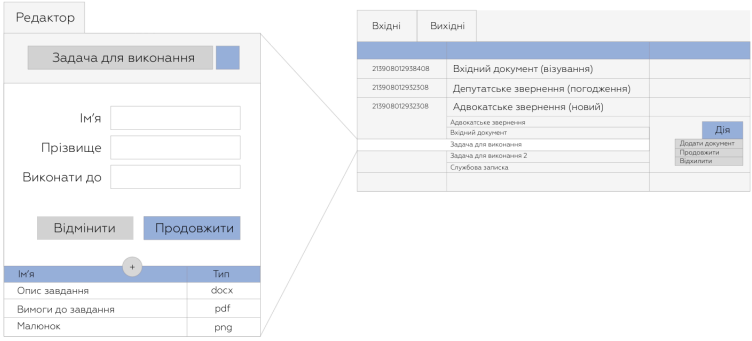

Звернення громадян
Загальний відділ
Керівництво міністерства
Сектор контролю
Структурний підрозділ виконавець
Вхідні документи
Ел. документи
Початок
Реєстрація
Керівництву
Так
Ні
Резолюції
Є задачі?
Ні
Так
Документ не має резолюцій, залишити його?
Ні
Так
Потребує контролю?
Ні
Так
Контроль
Виконання
Архів
Кінець

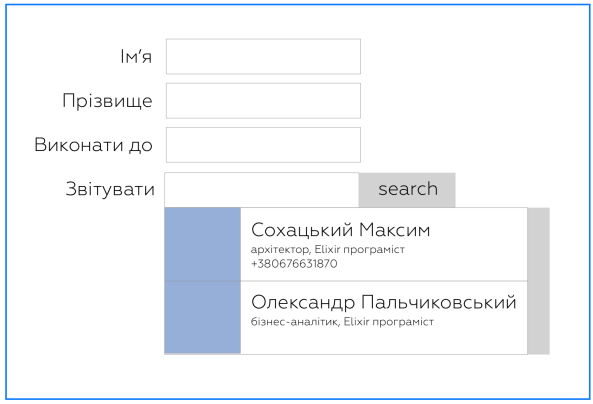

Лютий 2020
Нд Пн Вт Ср Чт Пт Сб
1
2 3 4 5 6 7 8
9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28 29

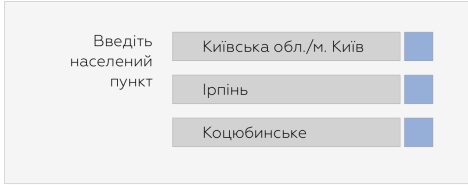

search
Сохацький Максим
архітектор, Elixir програміст
+380676631870
Олександр Пальчиковський
бізнес-аналітик, Elixir програміст

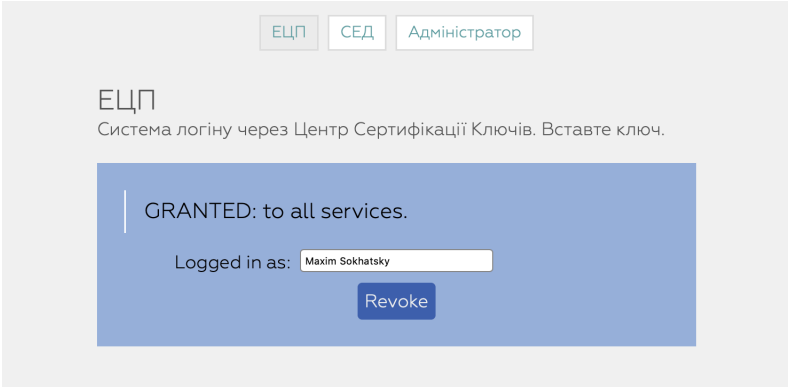

Редактор
Задача для виконання
Ім'я
Прізвище
Виконати до
Відмінити
Продовжити
+
Ім'я
Тип
Опис завдання
docx
Вимоги до завдання
pdf
Малюнок
png

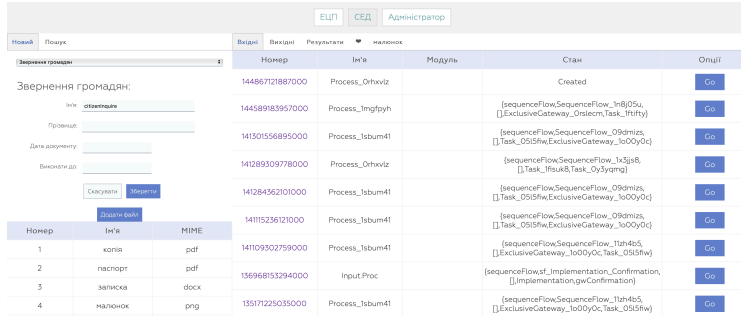

Вхідні
Вихідні
213908012938408
Вхідний документ (візування)
213908012932308
Депутатське звернення (погодження)
213908012932308
Адвокатське звернення (новий)
Адвокатське звернення
Вхідний документ
Задача для виконання
Задача для виконання 2
Службова записка
Дія
Додати документ
Продовжити
Відхилити

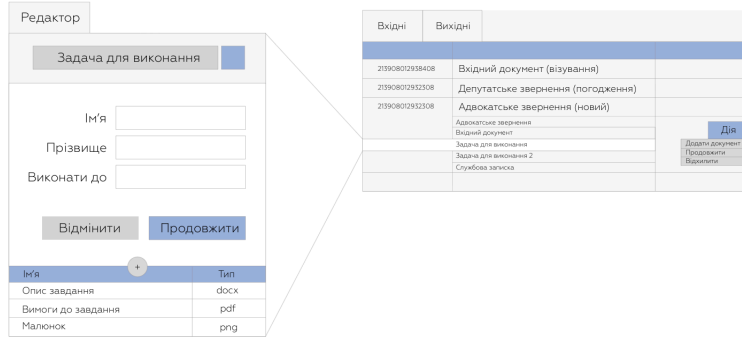

Редактор
Задача для виконання
Ім'я
Прізвище
Виконати до
Відмінити
Продовжити
Ім'я
Тип
Опис завдання
docx
Вимоги до завдання
pdf
Малюнок
png
Вхідні
Вихідні
213908012938408
Вхідний документ (візування)
213908012932308
Депутатське звернення (погодження)
213908012932308
Адвокатське звернення (новий)
Адвокатське звернення
Вхідний документ
Задача для виконання
Задача для виконання 2
Службова записка
Дія
Додати документ
Продовжити
Відхилити


Ім'я
Прізвище
Виконати до
Звітувати
search
Сохацький Максим
архітектор, Elixir програміст
+380676631870
Олександр Пальчиковський
бізнес-аналітик, Elixir програміст


Введіть населений пункт
Київська обл./м. Київ
Ірпінь
Коцюбинське


ЕЦП
СЕД
Адміністратор
ЕЦП
Система логіну через Центр Сертифікації Ключів. Вставте ключ.
GRANTED: to all services.
Logged in as:
Maxim Sokhatsky
Revoke


ЕЦП
СЕД
Адміністратор
Новий
Пошук
Вхідні
Вихідні
Результати
малюнок
Звернення громадян
Звернення громадян:
Ім'я
citizenInquire
Прізвище
Дата документу
Виконати до
Скасувати
Зберегти
Додати файл
Номер
Ім'я
MIME
1 копія pdf
2 паспорт pdf
3 записка docx
4 малюнок png
Номер
Ім'я
Модуль
Стан
Опції
1448671218870000 Process_0rhxvlz Created Go
1445891839570000 Process_1mgfpyh {sequenceFlow,SequenceFlow_1n8j05u,[],ExclusiveGateway_0rslecm,Task_1ftrifty} Go
1413011556895000 Process_1sbum41 {sequenceFlow,SequenceFlow_09dmizs,[],Task_0515flw,ExclusiveGateway_1o00y0c} Go
1412893097780000 Process_0rhxvlz {sequenceFlow,SequenceFlow_1x3jjsB,[],Task_1flsuk8,Task_0y3yqmg} Go
1412843621010000 Process_1sbum41 {sequenceFlow,SequenceFlow_09dmizs,[],Task_0515flw,ExclusiveGateway_1o00y0c} Go
1411152361210000 Process_1sbum41 {sequenceFlow,SequenceFlow_09dmizs,[],Task_0515flw,ExclusiveGateway_1o00y0c} Go
1411093027590000 Process_1sbum41 {sequenceFlow,SequenceFlow_11zh4b5,[],ExclusiveGateway_1o00y0c,Task_0515flw} Go
1369681532940000 Input.Proc {sequenceFlow,sf_Implementation_Confirmation,[],Implementation,gwConfirmation} Go
1351712250350000 Process_1sbum41 {sequenceFlow,SequenceFlow_11zh4b5,[],ExclusiveGateway_1o00y0c,Task_0515flw} Go


Редактор
Задача для виконання
Ім'я
Прізвище
Виконати до
Відмінити
Продовжити
Ім'я
Тип
Опис завдання docx
Вимоги до завдання pdf
Малюнок png
Вхідні
Вихідні
21390801293840 8 Вхідний документ (візування)
21390801293250 8 Депутатське звернення (погодження)
21390801293250 8 Адвокатське звернення (новий)
Адвокатське звернення
Вхідний документ
Задача для виконання
Задача для виконання 2
Службова записка
Дія
Додати документ
Продовжити
Відхилити

```
> :supervisor.which_children :n2o
[
  { {:ws, '/chat/ws/4'}, <0.985.0>, :worker, [:n2o_ws] },
  { {:ws, '/chat/ws/3'}, <0.984.0>, :worker, [:n2o_ws] },
  { {:ws, '/chat/ws/2'}, <0.983.0>, :worker, [:n2o_ws] },
  { {:ws, '/chat/ws/1'}, <0.982.0>, :worker, [:n2o_ws] },
  { {:mqtt, '/bpe/mqtt/4'}, <0.977.0>, :worker, [:n2o_mqtt] },
  { {:mqtt, '/bpe/mqtt/3'}, <0.976.0>, :worker, [:n2o_mqtt] },
  { {:mqtt, '/bpe/mqtt/2'}, <0.975.0>, :worker, [:n2o_mqtt] },
  { {:mqtt, '/bpe/mqtt/1'}, <0.974.0>, :worker, [:n2o_mqtt] },
  { {:caching, 'timer'}, <0.969.0>, :worker, [:n2o] }
]
```

```
> :kvs.all :writer
[
  {:writer, '/bpe/proc', 2},
  {:writer, '/erp/group', 1},
  {:writer, '/erp/partners', 7},
  {:writer, '/acc/synrc/Kyiv', 3},
  {:writer, '/chat/5HT', 1},
  {:writer, '/bpe/hist/1562187187807717000', 8},
  {:writer, '/bpe/hist/1562192587632329000', 1}
]
```

```
> erlc AuthenticationFramework.asn1
> erlc InformationFramework.asn1
> erlc KEP.asn1
```

```
> CA.CAdES.readSignature
[
  {:certinfo, ~c"TINUA-2955020254",
   "СОХАЦЬКИЙ МАКСИМ ЕРОТЕЙОВИЧ",
   "МАКСИМ ЕРОТЕЙОВИЧ", "СОХАЦЬКИЙ",
   "СОХАЦЬКИЙ МАКСИМ ЕРОТЕЙОВИЧ",
   [
     subjectKeyIdentifier: "VNXfTvJQccGtPgNhUftIQZV+mUROTgroLotsbtYZsFE=",
     authorityKeyIdentifier: "XphNUm+C84/0vi5ABGgN/rOvysLkBHVNB9CuTISwfB0=",
     keyUsage: [<<6, 192>>],
     certificatePolicies: {"https://acsk.privatbank.ua/acskdoc",
      ["1.2.804.2.1.1.1.2.2", "1.3.6.1.5.5.7.2.1"]},
     basicConstraints: [],
     qcStatements: {"https://acsk.privatbank.ua",
      ["0.4.0.1862.1.1", "0.4.0.1862.1.5", "1.3.6.1.5.5.7.11.2",
       "0.4.0.194121.1.1", "1.2.804.2.1.1.1.2.1"]},
     cRLDistributionPoints: ["http://acsk.privatbank.ua/crl/PB-2023-S6.crl"],
     freshestCRL: ["http://acsk.privatbank.ua/crldelta/PB-Delta-2023-S6.crl"],
     authorityInfoAccess: [
       {"1.3.6.1.5.5.7.48.2",
        "http://acsk.privatbank.ua/arch/download/PB-2023.p7b"},
       {"1.3.6.1.5.5.7.48.1", "http://acsk.privatbank.ua/services/ocsp/"}
     ],
     subjectInfoAccess: [
       {"1.3.6.1.5.5.7.48.3", "http://acsk.privatbank.ua/services/tsp/"}
     ],
     subjectDirectoryAttributes: [
       {"1.2.804.2.1.1.1.11.1.4.7.1", "0"},
       {"1.2.804.2.1.1.1.11.1.4.1.1", "2955020254"}
     ]
   ], "ФІЗИЧНА ОСОБА", "", "", ~c"UA", "КИЇВ"},
  {:certinfo, ~c"UA-14360570-2310",
   "КНЕДП АЦСК АТ КБ ПРИВАТБАНК\"\"", "", "",
   "КНЕДП АЦСК АТ КБ ПРИВАТБАНК\"\"",
   [
     contentType: "0.6.9.42.840.113549.1.7.1",
     signingTime: "240221110356Z",
     messageDigest: "MfvlhoDVCPkptQRN+S2zNGp0nrOsS93mLdbcz/kZ9GI=",
     signingCertificateV2: 54004158142501264913150880415587187613877419268,
     contentTimestamp: {"1.2.840.113549.1.7.2",
      369952533463044024072847521187489702 6, "20240221110626Z",
      "MfvlhoDVCPkptQRN+S2zNGp0nrOsS93mLdbcz/kZ9GI="}
   ], "АТ КБ ПРИВАТБАНК\"\"", "", "", ~c"UA", "Київ"}
]
```

```
defmodule CMS do
    def decrypt(cms, {schemeOID, privateKeyBin}) do
        {_,{:ContentInfo,_,{:EnvelopedData,_,_,x,y,_}}} = cms
        {:EncryptedContentInfo,_,{_,encOID,{_,<<_::16,iv::binary>>}},data} = y
              case :proplists.get_value(:kari,  x, []) do
        [] -> case :proplists.get_value(:ktri,  x, []) do
        [] -> case :proplists.get_value(:kekri, x, []) do
        [] -> case :proplists.get_value(:pwri,  x, []) do
        [] -> {:error, "Unknown Other Recepient Info"}
              pwri  -> pwri(pwri,   privateKeyBin, encOID, data, iv) end
              kekri -> kekri(kekri, privateKeyBin, encOID, data, iv) end
              ktri  -> ktri(ktri,   privateKeyBin, encOID, data, iv) end
              kari  -> kari(kari,   privateKeyBin, schemeOID, encOID, data, iv)
     end
    end
end
```

```
# openssl cms -decrypt -in encrypted.txt -inkey client.key -recip client.pem
# openssl cms -encrypt -aes256 -in message.txt -out encrypted.txt \
                       -recip client.pem -keyopt ecdh_kdf_md:sha256



def map(:'dhSinglePass-stdDH-sha512kdf-scheme'), do: :sha512
def map(:'dhSinglePass-stdDH-sha384kdf-scheme'), do: :sha384
def map(:'dhSinglePass-stdDH-sha256kdf-scheme'), do: :sha256
def eccCMS(ukm, bit), do:
    :'CMSECCAlgs-2009-02'.encode(:'ECC-CMS-SharedInfo', sharedInfo(ukm, bit))
def sharedInfo(ukm, len), do: {:'ECC-CMS-SharedInfo',
    {:'KeyWrapAlgorithm',{2,16,840,1,101,3,4,1,45},:asn1_NOVALUE}, ukm, <>}

def kari(kari, privateKeyBin, schemeOID, encOID, data, iv) do
    {_,:v3,{_,{_,_,publicKey}},ukm,{_,kdfOID,_},[{_,_,encryptedKey}]} = kari
    {scheme,_}  = CA.ALG.lookup(schemeOID)
    {kdf,_}     = CA.ALG.lookup(kdfOID)
    {enc,_}     = CA.ALG.lookup(encOID)
    sharedKey   = :crypto.compute_key(:ecdh,publicKey,privateKeyBin,scheme)
    {_,payload} = eccCMS(ukm, 256)
    derived     = KDF.derive(map(kdf), sharedKey, 32, payload)
    unwrap      = CA.AES.KW.unwrap(encryptedKey, derived)
    res         = CA.AES.decrypt(enc, data, unwrap, iv)
    {:ok, res}
end
def testDecryptECC(), do: CA.CMS.decrypt(testECC(), testPrivateKeyECC())

def testECC() do
    {:ok,base} = :file.read_file "priv/certs/encrypted.txt"
    [_,s] = :string.split base, "\n\n"
    x = :base64.decode s
    :'CryptographicMessageSyntax-2010'.decode(:ContentInfo, x)
end

def testPrivateKeyECC() do
    privateKey = :public_key.pem_entry_decode(pem("priv/certs/client.key"))
    {:'ECPrivateKey',_,privateKeyBin,{:namedCurve,schemeOID},_,_} = privateKey
    {schemeOID,privateKeyBin}
end
```

```
# openssl cms -encrypt -secretkeyid 07 \
              -secretkey 0123456789ABCDEF0123456789ABCDEF \
              -aes256 -in message.txt -out encrypted2.txt

# openssl cms -decrypt -in encrypted2.txt -secretkeyid 07 \
              -secretkey 0123456789ABCDEF0123456789ABCDEF
```

```
def kekri(kekri, privateKeyBin, encOID, data, iv) do
    {:'KEKRecipientInfo',_vsn,_,{_,kea,_},encryptedKey} = kekri
    _ = CA.ALG.lookup(kea)
    {enc,_} = CA.ALG.lookup(encOID)
    unwrap = CA.AES.KW.unwrap(encryptedKey,privateKeyBin)
    res = CA.AES.decrypt(enc, data, unwrap, iv)
    {:ok, res}
end
def testDecryptKEK(), do: CA.CMS.decrypt(testKEK(), testPrivateKeyKEK())

def testPrivateKeyKEK() do
    {:kek, :binary.decode_hex("0123456789ABCDEF0123456789ABCDEF")}
end

def testKEK() do
    {:ok,base} = :file.read_file "priv/certs/encrypted2.txt"
    [_,s] = :string.split base, "\n\n"
    x = :base64.decode s
    :'CryptographicMessageSyntax-2010'.decode(:ContentInfo, x)
end
```

```
# gpgsm --list-keys
# gpgsm --list-secret-keys
# gpgsm -r 0xD3C8F78A -e CNAME > cms.bin
# gpgsm -u 0xD3C8F78A -d cms.bin
# gpgsm --export-secret-key-p12 0xD3C8F78A > key.bin
# openssl pkcs12 -in key.bin -nokeys -out public.pem
# openssl pkcs12 -in key.bin -nocerts -nodes -out private.pem



def ktri(ktri, privateKeyBin, encOID, data, iv) do
    {:'KeyTransRecipientInfo',_vsn,_,{_,schemeOID,_},key} = ktri
    {:rsaEncryption,_} = CA.ALG.lookup schemeOID
    {enc,_} = CA.ALG.lookup(encOID)
    sessionKey = :public_key.decrypt_private(key, privateKeyBin)
    res = CA.AES.decrypt(enc, data, sessionKey, iv)
    {:ok, res}
end
def testDecryptRSA(), do: CA.CMS.decrypt(testRSA(), testPrivateKeyRSA())

def testPrivateKeyRSA() do
    {:ok,bin} = :file.read_file("priv/rsa-cms.key")
    pki = :public_key.pem_decode(bin)
    [{:PrivateKeyInfo,_,_}] = pki
    rsa = :public_key.pem_entry_decode(hd(pki))
    {:'RSAPrivateKey',:'two-prime',_n,_e,_d,_,_,_,_,_,_} = rsa
    {:rsaEncryption,rsa}
end

def testRSA() do
    {:ok,x} = :file.read_file "priv/rsa-cms.bin"
    :'CryptographicMessageSyntax-2010'.decode(:ContentInfo, x)
end







defmodule KDF do
    def hl(:md5),    do: 16
    def hl(:sha),    do: 20
    def hl(:sha224), do: 28
    def hl(:sha256), do: 32
    def hl(:sha384), do: 48
    def hl(:sha512), do: 64

    def derive(h, d, len, x) do
        :binary.part(:lists.foldr(fn i, a ->
            :crypto.hash(h, d <> <> <> x) <> a
        end, <<>>, :lists.seq(1,round(Float.ceil(len/hl(h))))), 0, len)
    end
end
```

```
-define(MSB64,      1/unsigned-big-integer-unit:64).
-define(DEFAULT_IV, << 16#A6A6A6A6A6A6A6A6:?MSB64 >>).

unwrap(CipherText, KEK) -> unwrap(CipherText, KEK, ?DEFAULT_IV).
unwrap(CipherText, KEK, IV)
        when (byte_size(CipherText) rem 8) =:= 0
        andalso (bit_size(KEK) =:= 128
            orelse bit_size(KEK) =:= 192
            orelse bit_size(KEK) =:= 256) ->
    BlockCount = (byte_size(CipherText) div 8) - 1,
    IVSize = byte_size(IV),
    case do_unwrap(CipherText, 5, BlockCount, KEK) of
        << IV:IVSize/binary, PlainText/binary >> ->
            PlainText;
        _ ->
            erlang:error({badarg, [CipherText, KEK, IV]})
    end.

codec(128) -> aes_128_ecb;
codec(192) -> aes_192_ecb;
codec(256) -> aes_256_ecb.

do_unwrap(Buffer, J, _BlockCount, _KEK) when J < 0 -> Buffer;
do_unwrap(Buffer, J, BlockCount, KEK) ->
    do_unwrap(do_unwrap(Buffer, J, BlockCount, BlockCount, KEK),
              J - 1, BlockCount, KEK).
do_unwrap(Buffer, _J, I, _BlockCount, _KEK) when I < 1 -> Buffer;
do_unwrap(<< A0:?MSB64, Rest/binary >>, J, I, BlockCount, KEK) ->
    HeadSize = (I - 1) * 8,
    << Head:HeadSize/binary, B0:8/binary, Tail/binary >> = Rest,
    Round = (BlockCount * J) + I,
    A1 = A0 bxor Round,
    Data = << A1:?MSB64, B0/binary >>,
    << A2:8/binary, B1/binary >>
        = crypto:crypto_one_time(codec(bit_size(KEK)),
          KEK, ?DEFAULT_IV, Data, [{encrypt,false}]),
    do_unwrap(<< A2/binary, Head/binary, B1/binary,
          Tail/binary >>, J, I - 1, BlockCount, KEK).
```

```
def decrypt(crypto_codec, data, key, iv \\ :crypto.strong_rand_bytes(16))
def decrypt(:'id-aes256-ECB',data,key,iv), do: decryptAES256ECB(data,key,iv)
def decrypt(:'id-aes256-CBC',data,key,iv), do: decryptAES256CBC(data,key,iv)
def decrypt(:'id-aes256-GCM',data,key,iv), do: decryptAES256GCM(data,key,iv)
def decrypt(:'id-aes256-CCM',data,key,iv), do: decryptAES256CCM(data,key,iv)
def test() do
  [
    check_SECP384R1_GCM256(),
    check_X25519_GCM256(),
    check_C2PNB368w1_GCM256(),
    check_BrainPoolP512t1_GCM256(),
    check_BrainPoolP512t1_GCM256(),
    check_SECT571_GCM256(),
    check_X448_GCM256(),
    check_X448_CBC256(),
    check_X448_ECB256(),
  ]
end
```

CMS-AES-CCM-and-AES-GCM-2009.asn1
CMSAesRsaesOaep-2009.asn1
CMSECCAlgs-2009-02.asn1
CMSECDHAlgs-2017.asn1
CryptographicMessageSyntax-2009.asn1
CryptographicMessageSyntax-2010.asn1
CryptographicMessageSyntaxAlgorithms-2009.asn1
EnrollmentMessageSyntax-2009.asn1
PKCS-10.asn1
PKCS-7.asn1
PKIX1Explicit-2009.asn1
PKIX1Implicit-2009.asn1
PKIXAlgs-2009.asn1
PKIXCMP-2009.asn1
PKIXCRMF-2009.asn1

```
def csr(user) do
     {ca_key, ca} = read_ca()
     priv = X509.PrivateKey.new_ec(:secp384r1)
     der = :public_key.der_encode(:ECPrivateKey, priv)
     pem = :public_key.pem_encode([{:ECPrivateKey, der, :not_encrypted}])
     :file.write_file(user <> ".key", pem)
     :io.format '~p~n', [priv]
     csr = X509.CSR.new(priv, "/C=UA/L=Kyiv/O=SYNRC/CN=" <> user,
        extension_request: [
           X509.Certificate.Extension.subject_alt_name(["n2o.dev"])])
     :io.format 'CSR: ~p~n', [csr]
     :file.write_file(user <> ".csr", X509.CSR.to_pem(csr))
     true = X509.CSR.valid?(csr)
     subject = X509.CSR.subject(csr)
     :io.format 'Subject ~p~n', [subject]
     :io.format 'CSR ~p~n', [csr]
     X509.Certificate.new(X509.CSR.public_key(csr), subject, ca, ca_key,
        extensions: [subject_alt_name:
          X509.Certificate.Extension.subject_alt_name(["n2o.dev", "erp.uno"])
     ])
     csr
 end
```

```
def ca() do
     ca_key = X509.PrivateKey.new_ec(:secp384r1)
     ca = X509.Certificate.self_signed(ca_key,
           "/C=UA/L=Kyiv/O=SYNRC/CN=CSR-CMP", template: :root_ca)
     der = :public_key.der_encode(:ECPrivateKey, ca_key)
     pem = :public_key.pem_encode([{:ECPrivateKey, der, :not_encrypted}])
     :file.write_file "ca.key", pem
     :file.write_file "ca.pem", X509.Certificate.to_pem(ca)
     {ca_key, ca}
 end

 def read_ca() do
     {:ok, ca_key_bin} = :file.read_file "ca.key"
     {:ok, ca_bin} = :file.read_file "ca.pem"
     {:ok, ca_key} = X509.PrivateKey.from_pem ca_key_bin
     {:ok, ca} = X509.Certificate.from_pem ca_bin
     {ca_key, ca}
 end



 def server(name) do
     {ca_key, ca} = read_ca()
     server_key = X509.PrivateKey.new_ec(:secp384r1)
       X509.Certificate.new(X509.PublicKey.derive(server_key),
          "/C=UA/L=Kyiv/O=SYNRC/CN=" <> name, ca, ca_key,
          extensions: [subject_alt_name:
             X509.Certificate.Extension.subject_alt_name(["n2o.dev", "erp.uno"
     ]) ])
 end
```

```
defmodule CA.CMP do
  @moduledoc "CA/CMP TCP server."
  require CA

  def start(), do: :erlang.spawn(fn -> listen(1829) end)
  def listen(port) do
      {:ok, socket} = :gen_tcp.listen(port,
        [:binary, {:packet, 0}, {:active, false}, {:reuseaddr, true}])
      accept(socket)
  end

  def accept(socket) do
      {:ok, fd} = :gen_tcp.accept(socket)
      :erlang.spawn(fn -> __MODULE__.loop(fd) end)
      accept(socket)
  end

  def loop(socket) do
      case :gen_tcp.recv(socket, 0) do
           {:ok, data} ->
               [headers,body] = :string.split data, "\r\n\r\n", :all
               {:ok,dec} = :'PKIXCMP-2009'.decode(:'PKIMessage', body)
               {:PKIMessage, header, body, code, _extra} = dec
               __MODULE__.message(socket, header, body, code)
               loop(socket)
          {:error, :closed} -> :exit
      end
  end
```

```
 def answer(socket, header, body, code) do
      message = CA."PKIMessage"(header: header, body: body, protection: code)
      {:ok, bytes} = :'PKIXCMP-2009'.encode(:'PKIMessage', message)
      res =  "HTTP/1.0 200 OK\r\n"
          <> "Server: SYNRC CA/CMP\r\n"
          <> "Content-Type: application/pkixcmp\r\n\r\n"
          <> :erlang.iolist_to_binary(bytes)
      :gen_tcp.send(socket, res)
  end
```

```
. iex -S mix
> CA.CSR.ca
> CA.CSR.csr "maxim"
```

```
# openssl cmp -cmd p10cr -server localhost:1829 \
#             -path . -srvcert ca.pem -ref cmptestp10cr \
#             -secret pass:0000 -certout . client.csr
```

```
def message(socket, header, {:p10cr, csr} = body, code) do
      {:PKIHeader, pvno, from, to, messageTime, protectionAlg,
         _senderKID, _recipKID, transactionID, senderNonce,
         _recipNonce, _freeText, _generalInfo} = header
      true = code == validateProtection(header, body, code)

      {ca_key, ca} = CA.CSR.read_ca()
      subject = X509.CSR.subject(csr)
      :io.format '~p~n',[subject]
      true = X509.CSR.valid?(CA.parseSubj(csr))
      cert = X509.Certificate.new(X509.CSR.public_key(csr),
         CA.CAdES.subj(subject), ca, ca_key,
         extensions: [subject_alt_name:
            X509.Certificate.Extension.subject_alt_name(["synrc.com"]) ])

      reply = CA."CertRepMessage"(response:
            [ CA."CertResponse"(certReqId: 0,
              certifiedKeyPair: CA."CertifiedKeyPair"(certOrEncCert:
                {:certificate, {:x509v3PKCert, CA.convertOTPtoPKIX(cert)}}),
              status: CA."PKIStatusInfo"(status: 0))])

      pkibody = {:cp, reply}
      pkiheader = CA."PKIHeader"(sender: to, recipient: from, pvno: pvno,
          recipNonce: senderNonce, transactionID: transactionID,
          protectionAlg: protectionAlg, messageTime: messageTime)
      answer(socket, pkiheader, pkibody,
          validateProtection(pkiheader, pkibody, code))
  end
```

```
def message(socket, header, {:certConf, statuses}, code) do
      {:PKIHeader, _, from, to, _, _, _, _, _, senderNonce, _, _, _} = header

      :lists.map(fn {:CertStatus,bin,no,{:PKIStatusInfo, :accepted, _, _}} ->
          :logger.info 'CERTCONF ~p request ~p~n', [no,:binary.part(bin,0,8)]
      end, statuses)

      pkibody = {:pkiconf, :asn1_NOVALUE}
      pkiheader = CA."PKIHeader"(header, sender: to, recipient: from,
          recipNonce: senderNonce)
      answer(socket, pkiheader, pkibody,
          validateProtection(pkiheader, pkibody, code))
  end
```

```
CMP info: sending P10CR
CMP info: received CP
CMP info: sending CERTCONF
CMP info: received PKICONF
CMP info: received 1 enrolled certificate(s), saving to file 'maxim.pem'
```

```
# openssl cmp -cmd genm -server 127.0.0.1:1829 \
#             -recipient "/CN=CMPserver" -ref 1234 -secret pass:0000

def message(_socket, _header, {:genm, req} = _body, _code) do
      :io.format 'generalMessage: ~p~n', [req]
  end
```

```
# openssl cmp -cmd ir -server 127.0.0.1:1829 \
#             -path . -srvcert ca.pem -ref NewUser \
#             -secret pass:0000 -certout maxim.pem \
#             -newkey maxim.key -subject "/CN=maxim/O=SYNRC/ST=Kyiv/C=UA"
```

```
 def message(_socket, _header, {:ir, req}, _) do
      :lists.map(fn {:CertReqMsg, req, sig, code} ->
         :io.format 'request: ~p~n', [req]
         :io.format 'signature: ~p~n', [sig]
         :io.format 'code: ~p~n', [code]
      end, req)
  end
```

```
# openssl cmp -cmd cr -server 127.0.0.1:1829 \
#             -path . -srvcert ca.pem -ref NewUser \
#             -secret pass:0000 -certout maxim.pem \
#             -newkey maxim.key -subject "/CN=maxim/O=SYNRC/ST=Kyiv/C=UA"
```

```
defmodule CA do
  use Application
  use Supervisor

  require Record
  Enum.each(Record.extract_all(from_lib: "ca/include/PKIXCMP-2009.hrl"),
            fn {name, definition} -> Record.defrecord(name, definition) end)

  Enum.each(Record.extract_all(from_lib: "public_key/include/public_key.hrl"),
            fn {name, definition} -> Record.defrecord(name, definition) end)

  def init([]), do: {:ok, { {:one_for_one, 5, 10}, []} }
  def start(_type, _args) do
      :logger.add_handlers(:ldap)
      CA.CMP.start
      CA.CMC.start
      :supervisor.start_link({:local, __MODULE__}, __MODULE__, [])
  end
```

```
# apt install elixir
```

```
defmodule LDAP.Mixfile do
  use Mix.Project
  def project(), do:
    [ app: :ldap, version: "8.7.20", deps: deps(),
      releases: [ ldap: [ include_executables_for: [:unix],
                          cookie: "SYNRC:LDAP" ] ] ]
  def application(), do:
    [ mod: {LDAP, []},
      extra_applications: [ :eldap, :asn1 ] ] end
  def deps(), do:
    [ {:exqlite, "~> 0.13.14"} ]
end
```

$$,5/,,,,.!$$

```
# erlc LADP.asn1
```

```
defmodule DS do
    require Record
    Enum.each(Record.extract_all(from_lib: "ldap/include/LDAP.hrl"),
              fn {name, definition} -> Record.defrecord(name, definition) end)
end



defmodule LDAP do
   import Exqlite.Sqlite3
   require DS
   use Application
   use Supervisor
   def code(), do: :binary.encode_hex(:crypto.strong_rand_bytes(8))
   def init([]), do: {:ok, { {:one_for_one, 5, 10}, []} }
   def start(_, _) do
       :logger.add_handlers(:ldap)
       :supervisor.start_link({:local, LDAP}, LDAP, [])
   end
   def initDB(path) do
       {:ok, conn} = open(path)
       :logger.info 'SYNRC LDAP Instance: ~p', [path]
       :logger.info 'SYNRC LDAP Connection: ~p', [conn]
       execute(conn, "create table ldap (rdn text,att text,val binary)")
       :ok = execute(conn, "PRAGMA journal_mode = OFF;")
       :ok = execute(conn, "PRAGMA temp_store = MEMORY;")
       :ok = execute(conn, "PRAGMA cache_size = 1000000;")
       :ok = execute(conn, "PRAGMA synchronous = 0;")
       conn
   end
   def listen(port,path) do
       conn = initDB(path)
       {:ok, socket} = :gen_tcp.listen(port,
         [:binary, {:packet, 0}, {:active, false}, {:reuseaddr, true}])
       accept(socket,conn)
   end
   def accept(socket,conn) do
       {:ok, fd} = :gen_tcp.accept(socket)
       :erlang.spawn(fn -> loop(fd, conn) end)
       accept(socket,conn)
   end
```

```
    def start() do
        :erlang.spawn(fn ->
            listen(:application.get_env(:ldap,:port,1489),
                   :application.get_env(:ldap,:instance,code())) end)
    end
    def answer(response, no, op, socket) do
        message = DS."LDAPMessage"(messageID: no, protocolOp: {op, response})
        {:ok, bytes} = :'LDAP'.encode(:'LDAPMessage', message)
        send = :gen_tcp.send(socket, :erlang.iolist_to_binary(bytes))
    end
    def loop(socket, db) do
        case :gen_tcp.recv(socket, 0) do
             {:ok, data} ->
                  case :'LDAP'.decode(:'LDAPMessage',data) do
                       {:ok,decoded} ->
                           {:'LDAPMessage', no, payload, _} = decoded
                           message(no, socket, payload, db)
                           loop(socket, db)
                       {:error,reason} ->
                          :logger.error 'ERROR: ~p', [reason]
                         :exit
                  end
             {:error, :closed} -> :exit
        end
    end
end



# mix deps.get
# iex -S mix
> LDAP.start
#PID<0.311.0>
iex(2)>
04:58:26.030 [info]  SYNRC LDAP Instance: "416C4C41ED2C7060"
04:58:26.030 [info]  SYNRC LDAP Connection: #Reference<
     0.1146704550.396492828.212314>
iex(3)>
nil
```

```
createDN(conn, "dc=synrc,dc=com",
    [ attr("dc",["synrc"]), attr("objectClass",["top","domain"]) ])
createDN(conn, "ou=schema",
    [ attr("ou",["schema"]), attr("objectClass",["top","domain"]) ])
createDN(conn, "cn=tonpa,dc=synrc,dc=com",
    [ attr("cn",["tonpa"]),attr("uid",["1000"]),
      attr("objectClass",["inetOrgPerson","posixAccount"]) ])
createDN(conn, "cn=rocco,dc=synrc,dc=com",
    [ attr("cn",["rocco"]),attr("uid",["1001"]),
      attr("objectClass",["inetOrgPerson","posixAccount"]) ])
createDN(conn, "cn=admin,dc=synrc,dc=com",
    [ attr("rootpw",["secret"]), attr("cn",["admin"]),
      attr("objectClass",["inetOrgPerson"]) ])

def appendNotEmpty([]),  do: []
def appendNotEmpty(res) do
    res ++ case res do [] -> [] ; _ -> ',' end
end
def createDN(db, dn, attributes) do
    norm  = :lists.foldr(fn {:PartialAttribute, att, vals}, acc ->
            :lists.map(fn val -> [qdn(dn),att,val] end, vals) ++
                                acc end, [], attributes)
    {_,p} = :lists.foldr(fn x, {acc,res} ->
             {acc + length(x), appendNotEmpty(res)
              ++ :io_lib.format('(?~p,?~p,?~p)', [acc+1,acc+2,acc+3])}
            end, {0,[]}, norm)
    {:ok, statement} = prepare(db,
          'insert into ldap (rdn,att,val) values ' ++ p ++ '')
    :ok = bind(db, statement, :lists.flatten(norm))
    :done = step(db, statement)
end
def message(no, socket, {:bindRequest, {_,_,bindDN,{:simple, password}}}, db)
   do
    sql = "select rdn, att from ldap where " <>
          "rdn = ?1 and att = 'rootpw' and val = ?2"
    {:ok, statement} = prepare db, sql
    bind(db, statement, [hash(qdn(bindDN)),password])
    case step(db, statement) do
        :done ->  code = :invalidCredentials
                  :logger.error 'BIND Error: ~p', [code]
                  response = DS."BindResponse"(resultCode: code,
                      matchedDN: "", diagnosticMessage: 'ERROR')
                  answer(response, no, :bindResponse, socket)
         {:row,[dn,password]} ->
                  :logger.info 'BIND DN: ~p', [bindDN]
                  response = DS."BindResponse"(resultCode: :success,
                       matchedDN: "", diagnosticMessage: 'OK')
                  answer(response, no, :bindResponse, socket)
    end
end
def message(no, socket, {:bindRequest, {_,_,bindDN,creds}}, db) do
    code = :authMethodNotSupported
    :logger.info 'BIND ERROR: ~p', [code]
    response = DS."BindResponse"(resultCode: code,
       matchedDN: "", diagnosticMessage: 'ERROR')
    answer(response, no, :bindResponse, socket)
end
```

```
def message(no, socket, {:addRequest, {_,dn, attributes}}, db) do
      {:ok, statement} = prepare(db, "select rdn, att, val from ldap where rdn =
       ?1")
      bind(db, statement, [hash(qdn(dn))])
      case step(db, statement) do
            {:row, _} ->
                :logger.info 'ADD ERROR: ~p', [dn]
                resp = DS.'LDAPResult'(resultCode: :entryAlreadyExists,
                       matchedDN: dn, diagnosticMessage: 'ERROR')
                answer(resp, no, :addResponse, socket)
            :done ->
                createDN(db, dn, attributes)
                :logger.info 'ADD DN: ~p', [dn]
                resp = DS.'LDAPResult'(resultCode: :success,
                       matchedDN: dn, diagnosticMessage: 'OK')
                answer(resp, no, :addResponse, socket)
       end
   end
```

```
def attr(k,v),      do: {:PartialAttribute, k, v}
   def node(dn,attrs), do: {:SearchResultEntry, dn, attrs}

   def message(no, socket,
       {:searchRequest, {_,"",scope,_,limit,_,_,filter,attributes}}, db) do

       :logger.info 'DSE Scope: ~p', [scope]
       :logger.info 'DSE Filter: ~p', [filter]
       :logger.info 'DSE Attr: ~p', [attributes]

       :lists.map(fn response -> answer(response,no,:searchResEntry,socket) end,
         [ node("", [
              attr("supportedLDAPVersion", ['3']),
              attr("namingContexts", ['dc=synrc,dc=com','ou=schema']),
              attr("supportedControl", ['1.3.6.1.4.1.4203.1.10.1']),
              attr("supportedExtensions", ['1.3.6.1.4.1.4203.1.11.3']),
              attr("altServer", ['ldap.synrc.com']),
              attr("subschemaSubentry", ['ou=schema']),
              attr("vendorName", ['SYNRC LDAP']),
              attr("vendorVersion", ['2.0']),
              attr("supportedSASLMechanisms", ['SIMPLE']),
              attr("objectClass", ['top','extensibleObject']),
              attr("entryUUID", [code()])]),
              attr("supportedFeatures", [ '1.3.6.1.1.14',
                                          '1.3.6.1.4.1.4203.1.5.1'])
         ])

       resp = DS.'LDAPResult'(resultCode: :success, matchedDN: "",
      diagnosticMessage: 'OK')
       answer(resp, no, :searchResDone,socket)
   end
```

```
def modifyDN(db, dn, attributes), do:
    :lists.map(fn {_, :add, x}     -> modifyAdd(db,dn,x)
                  {_, :replace, x} -> modifyReplace(db,dn,x)
                  {_, :delete, x}  -> modifyDelete(db,dn,x) end, attributes)

def modifyAdd(db, dn, {_,att,[val]}) do
    {:ok, st} = prepare(db, "insert into ldap (rdn,att,val) values (?1,?2,?3)
   ")
    :logger.info 'MOD ADD RDN: ~p', [hash(qdn(dn))]
    bind(db, st, [hash(qdn(dn)),att,val])
    step(db,st)
end

def modifyReplace(db, dn, {_,att,[val]}) do
    {:ok, st} = prepare(db, "update ldap set val = ?1 where rdn = ?2 and att
   = ?3")
    :logger.info 'MOD REPLACE RDN: ~p', [hash(qdn(dn))]
    bind(db, st, [val,hash(qdn(dn)),att])
    step(db,st)
end

def modifyDelete(db, dn, {_,att,_}) do
    {:ok, st} = prepare(db, "delete from ldap where rdn = ?1 and att = ?2")
    :logger.info 'MOD DEL RDN: ~p', [hash(qdn(dn))]
    bind(db, st, [hash(qdn(dn)),att])
    res = step(db,st)
    collect0(db,st,res,[])
end

def message(no, socket, {:modifyRequest, {_,dn, attributes}}, db) do
   {:ok, statement} = prepare(db, "select rdn, att, val from ldap where rdn =
    ?1")
   bind(db, statement, [hash(qdn(dn))])
   case step(db, statement) do
        {:row, _} -> :logger.info 'MOD DN: ~p', [dn]
                     modifyDN(db, dn, attributes)
                     resp = DS.'LDAPResult'(resultCode: :success,
                         matchedDN: dn, diagnosticMessage: 'OK')
                     answer(resp, no, :modifyResponse, socket)
        :done ->     :logger.info 'MOD ERROR: ~p', [dn]
                     resp = DS.'LDAPResult'(resultCode: :noSuchObject,
                        matchedDN: dn, diagnosticMessage: 'ERROR')
                     answer(resp, no, :modifyResponse, socket)
   end
end
```

```
    def modifyRDN(socket, no, db, dn, new, del) do
        {:ok, st} = prepare(db, "update ldap set rdn = ?1 where rdn = ?2")
        :logger.info 'MODIFY RDN UPDATE: ~p', [hash(qdn(dn))]
        bind(db, st, [new,hash(qdn(dn))])
        step(db,st)
    end

    def message(no, socket, {:modDNRequest, {_,dn,new,del,_}}, db) do
        :logger.info 'MOD RDN DN: ~p', [dn]
        :logger.info 'MOD RDN newRDN: ~p', [new]
        :logger.info 'MOD RDN deleteOldRDN: ~p', [del]
        modifyRDN(socket, no, db, dn, new, del)
        resp = DS.'LDAPResult'(resultCode: :success,
            matchedDN: dn, diagnosticMessage: 'OK')
        answer(resp, no, :modDNResponse, socket)
    end
```

```
def deleteDN(db, dn) do
        {:ok, st} = prepare(db, "delete from ldap where rdn = ?1")
        bind(db, st, [hash(qdn(dn))])
        res = step(db,st)
        collect0(db,st,res,[])
    end

    def message(no, socket, {:delRequest, dn}, db) do
        :logger.info 'DEL DN: ~p', [dn]
        deleteDN(db, dn)
        resp = DS.'LDAPResult'(resultCode: :success, matchedDN: dn,
       diagnosticMessage: 'OK')
        answer(resp, no, :delResponse, socket)
    end
```

```
def message(no, socket, {:compareRequest, {_,dn, assertion}}, db) do
        :logger.info 'CMP DN: ~p', [dn]
        :logger.info 'CMP Assertion: ~p', [assertion]
        result = compareDN(db, db, assertion)
        resp = DS.'LDAPResult'(resultCode: :success, matchedDN: dn,
      diagnosticMessage: 'OK')
        answer(resp, no, :compareResponse, socket)
    end
```

```
    def message(no, socket, {:abandonRequest, _}, db), do: :gen_tcp.close(socket)
    def message(no, socket, {:unbindRequest,  _}, db), do: :gen_tcp.close(socket)
```

AESKeyWrapWithPad-02.asn1
AESKeyWrapWithPad-88.asn1
ANSI-X9-42.asn1
ANSI-X9-62.asn1
AlgorithmInformation-2009.asn1
AttributeCertificateVersion1-2009.asn1
AuthenticationFramework.asn1
BasicAccessControl.asn1
CHAT.asn1
CMS-AES-CCM-and-AES-GCM-2009.asn1
CMSAesRsaesOaep-2009.asn1
CMSECCAlgs-2009-02.asn1
CMSECDHAlgs-2017.asn1
CertificateExtensions.asn1
Character-Coding-Attributes.asn1
Character-Presentation-Attributes.asn1
Character-Profile-Attributes.asn1
Colour-Attributes.asn1
CryptographicMessageSyntax-2009.asn1
CryptographicMessageSyntax-2010.asn1
CryptographicMessageSyntaxAlgorithms-2009.asn1
DOR-definition.asn1
Default-Value-Lists.asn1
DirectoryAbstractService.asn1
Document-Profile-Descriptor.asn1
EnrollmentMessageSyntax-2009.asn1
ExtendedSecurityServices-2009.asn1
External-References.asn1
Geo-Gr-Coding-Attributes.asn1
Geo-Gr-Presentation-Attributes.asn1
Geo-Gr-Profile-Attributes.asn1
ISO-STANDARD-9541-FONT-ATTRIBUTE-SET.asn1
ISO9541-SN.asn1
Identifiers-and-Expressions.asn1
InformationFramework.asn1
KEP.asn1
LDAP.asn1
Layout-Descriptors.asn1
Link-Descriptors.asn1
Location-Expressions.asn1
Logical-Descriptors.asn1
MultipleSignatures-2010.asn1
OCSP.asn1
PKCS-10.asn1
PKCS-12.asn1
PKCS-5.asn1
PKCS-7.asn1
PKCS-8.asn1
PKCS-9.asn1
PKIX-CommonTypes-2009.asn1
PKIX-X400Address-2009.asn1
PKIX1-PSS-OAEP-Algorithms-2009.asn1
PKIX1Explicit-2009.asn1
PKIX1Explicit88.asn1
PKIX1Implicit-2009.asn1

PKIX1Implicit88.asn1
PKIXAlgs-2009.asn1
PKIXAttributeCertificate-2009.asn1
PKIXCMP-2009.asn1
PKIXCRMF-2009.asn1
Raster-Gr-Coding-Attributes.asn1
Raster-Gr-Presentation-Attributes.asn1
Raster-Gr-Profile-Attributes.asn1
SMIMESymmetricKeyDistribution-2009.asn1
SecureMimeMessageV3dot1-2009.asn1

$+ +21.$