

Selective Identity Disclosure

Criptografia Aplicada

Diogo Almeida nº108902



Mestrado Cibersegurança

Professores: André Zúquete e Tomás Silva

23 de dezembro de 2024

Capítulo 1

Introdução

O mundo digital tem estado em constante evolução, e garantir a segurança e a privacidade é um ponto crucial nos dias de hoje. Os métodos tradicionais de identificação, como o cartão de cidadão, apresentam desafios em relação à privacidade dos dados.

O Selective Identity Disclosure (SID) apresenta uma abordagem ao problema de privacidade do cartão de cidadão, permitindo aos utilizadores esconder atributos do cartão, mas mantendo a garantia da autenticidade dos dados. Isto é possível através do uso de um DCC (Digital Citizen Card), um cartão de cidadão digital assinado por uma entidade superior, que garante a autenticidade do mesmo.

Neste projeto, foram desenvolvidas quatro aplicações: uma para criar o cartão digital, uma para que uma entidade superior assine o cartão, uma para esconder atributos do cartão e, por fim, uma para verificar a autenticidade dos dados e garantir que o cartão está íntegro.

Capítulo 2

req_dcc

2.0.1 Ler Dados e gerar pedido

O passo inicialmente implementado foi a leitura do cartão de cidadão utilizando a biblioteca **PyKCS11** do python. Com este passo são retirados os seguintes dados do cartão de cidadão do owner: nome, cc number, pais, organização e data de nascimento. Os atributos são extraídos do certificado do cartão de cidadão do owner.

Listing 2.1: Exemplo dos dados retirados

```
1 if isinstance(cert_value, bytes):
2     # Parse the certificate
3     cert = load_der_x509_certificate(cert_value)
4     subject = cert.subject
5     # Extract details
6     full_name = subject.get_attributes_for_oid(NameOID.COMMON_NAME)[0].value
7     id_number = subject.get_attributes_for_oid(NameOID.SERIAL_NUMBER)[0].value
8
9     # Additional data (may require custom OIDs for NIF, NSS, Utent, Birth Date)
10    country = (
11        subject.get_attributes_for_oid(NameOID.COUNTRY_NAME)[0].value
12        if subject.get_attributes_for_oid(NameOID.COUNTRY_NAME) else None
13    )
14    organization = (
15        subject.get_attributes_for_oid(NameOID.ORGANIZATION_NAME)[0].value
16        if subject.get_attributes_for_oid(NameOID.ORGANIZATION_NAME) else None
17    )
```

A data de nascimento tem de ser retirada das "extensions" do certificado utilizando depois o OID do atributo da data de nascimento.

Listing 2.2: Retirar data de nascimento

```
1 for ex in cert.extensions:
2     oid = ex.oid.dotted_string
3     if oid == "2.5.29.9": # subjectDirectoryAttributes OID
4         birth_date = str(get_birth_date_from_extension(ex.value.value))
5
6 def get_birth_date_from_extension(extension):
7     try:
8         decoded_value, _ = decode(extension)
9         for seq in decoded_value:
10             if isinstance(seq, Sequence):
11                 oid = str(seq[0]) # Extract the OID
12                 if oid == "1.3.6.1.5.5.7.9.1": # Birth Date OID
13                     # Extract the associated value
14                     birth_date_raw = seq[1][0] # Get the SetOf value
15                     birth_date_str = birth_date_raw.asOctets().decode() # Convert
16                                     to string
17
18                                     # Convert the string to a datetime object
```

```

18         birth_date = datetime.strptime(birth_date_str, "%Y%m%d%H%M%S")
19
20         # Set timezone to UTC (as 'Z' in the string indicates UTC)
21         birth_date = birth_date.replace(tzinfo=timezone.utc)
22
23         return birth_date
24
25     return None
26 except Exception as e:
27     print(f"Error decoding subjectDirectoryAttributes: {e}")
28     return None

```

Depois é pedida uma senha secreta ao owner. Com esta senha é gerado uma máscara para cada atributo utilizando o algoritmo SHA256. Com a máscara e o respetivo valor do atributo é então gerado um valor de commitment utilizando SHA256.

É retirado do cartão de cidadão do utilizador a chave pública do owner e colocada também no pedido.json. O pedido final contém os atributos retirados do cartão, os valores de commitment e a chave pública do owner.

Listing 2.3: JSON do pedido

```

1 {
2   "attributes": [
3     {
4       "label": "nome",
5       "value": "Owner name",
6       "commitment": "9
          bff8165f1abf93c99bec5dd17bcda5f9f8bb6e3cc8d6d05577a9b1a2279f9d9"
7     },
8     {
9       "label": "morada",
10      "value": "Rua",
11      "commitment": "
          eb8efaed4cc544df68730bae65293ca74e8ba448cdd9b893fd94b96fb819e7a4"
12    },
13    {
14      "label": "data_nascimento",
15      "value": "01/01/2000",
16      "commitment": "114
          b6d2001154bb8268584ed54d21bf394465285302117c003a78709a4135dd5"
17    },
18    {
19      "label": "cc_number",
20      "value": "number"
21      "commitment": "
          fee113691dcce3a8ee4a6f2368acc0c8a1d93dfea98abc845053d7cdd2dc137c"
22    }
23  ],
24  "digest_function": {
25    "type": "SHA-256"
26  },
27  "chave_publica_owner": [
28    {
29      "value": "-----BEGIN PUBLIC KEY-----\nMIIBIjWHPdgIg2s5XEjLCJO+gwTfe\nZQAB
          \n-----END PUBLIC KEY-----\n"
30      "description of key nature": "some description of the nature of the
          asymmetric cryptosystem of this key"
31    }
32  ]
33 }

```

De seguida a aplicação possui sockets para enviar este pedido para a aplicação do issuer para ser assinada por o mesmo.

Quando recebe um pedido assinado por um issuer, esta aplicação possui uma função que vai ler o certificado e ler os dados e verificar se a assinatura e os dados estão corretos de forma a garantir que os dados estão íntegros e que os dados assinados pelo issuer não foram alterados.

Listing 2.4: Verificar integridade dos dados

```
1  compromissos_e_chave_publica = []
2      for atributo in resposta["attributes"]:
3          compromissos_e_chave_publica.append(atributo["commitment"])
4
5      compromissos_e_chave_publica.append(resposta["chave_publica_owner"][0]["
        value"])
6
7      # Gerar os dados para assinar
8      dados_para_assinar = json.dumps(compromissos_e_chave_publica, sort_keys=True
        )
9      print("Dados para assinar:", dados_para_assinar)
10
11     hash_dados = hashes.Hash(hashes.SHA1(), backend=default_backend())
12     hash_dados.update(dados_para_assinar.encode())
13     digest = hash_dados.finalize()
14     # Verificar a assinatura
15
16     chave_publica.verify(
17         assinatura,
18         digest,
19         ec.ECDSA(hashes.SHA1())
20     )
21
22     print("Assinatura valida: os dados estao integros.")
23     return True
```

Capítulo 3

gen_dcc

3.0.1 Receber um pedido e assinar

Primeiramente foram gerados uma chave privada e um certificado. Para gerar isto foi utilizado o sistema curvas elípticas do OpenSSL.

Listing 3.1: Gerar certificado e chave privada

```
1 openssl ecparam -name prime256v1 -genkey -noout -out chave_privada_ec.pem
2
3 openssl req -new -x509 -key chave_privada_ec.pem -out certificado_autoassinado.pem -
  days 365
```

O gen_dcc utiliza sockets e fica sempre à espera de um JSON de pedido. Ao receber um pedido é carregada a chave privada gerada anteriormente,

Listing 3.2: Carregar chave privada

```
1 with open("chave_privada_ec.pem", "rb") as f:
2     chave_privada_issuer = serialization.load_pem_private_key(
3         f.read(),
4         password=None,
5         backend=default_backend()
6     )
```

São então lidos todos os commitments e a chave pública do owner presente no pedido JSON e estes valores são assinados com a chave privada do issuer.

Listing 3.3: Dados para assinar

```
1 compromissos_e_chave_publica = []
2     for atributo in pedido_dcc["attributes"]:
3         compromissos_e_chave_publica.append(atributo["commitment"])
4
5     compromissos_e_chave_publica.append(pedido_dcc["chave_publica_owner"][0]["value"
6     ])
7
8     # Gerar os dados para assinar
9     dados_para_assinar = json.dumps(compromissos_e_chave_publica, sort_keys=True)
10    assinatura_issuer = assinar_dados(chave_privada_issuer, dados_para_assinar)
```

Os dados são assinados utilizando o algoritmo SHA1 para não utilizar o mesmo algoritmo utilizado no req_dcc. São utilizadas curvas elípticas para fazer a assinatura.

Listing 3.4: Assinar dados

```

1 def assinar_dados(chave_privada, dados):
2     # Calcula o hash dos dados
3     hash_dados = hashes.Hash(hashes.SHA1(), backend=default_backend())
4     hash_dados.update(dados.encode())
5     digest = hash_dados.finalize()
6
7     # Assina os dados com a chave privada
8     assinatura = chave_privada.sign(
9         digest,
10        ec.ECDSA(hashes.SHA1()))
11
12    return assinatura

```

É então adicionado ao pedido JSON mais um campo que contém a assinatura dos dados, o timestamp da assinatura, uma descrição da assinatura e o certificado do issuer.

Listing 3.5: Novo campo do JSON

```

1 "Issuer_signature_over_comments_and_public_key": [
2     {
3         "value": "304502204d7b2e9190a4c87a967996cbe997218bf",
4         "timestamp": 1734465734.8392873,
5         "description": "Assinatura com Elliptic Curve (ECDSA) e SHA-256",
6         "issuer_certificate": "-----BEGIN CERTIFICATE-----\n
          nMIIBoTCCAXegAwIBAgIUfn4Iv91THVrwq1C0oHgww0tZIVcwCgYIKoZIzj0EAwIw\
          nPjELMAkGA1UEBhMCUFQxDzANBgNVBAGMBkF2ZWlybzEPMA0GA1UEBwwGQXZ\n
          END CERTIFICATE-----\n"
7     }
8 ]

```

É então enviado o DCC final para o owner e este guarda-o num ficheiro json.

Capítulo 4

gen_min_dcc

4.0.1 Esconder atributos do DCC

Nesta aplicação, primeiramente é pedido ao owner um DCC em formato JSON. Caso o utilizador passe um DCC com formato válido são então apresentados ao owner todos os atributos presentes no mesmo. O owner pode então escolher os atributos que pretende apresentar no min_dcc final de forma a não ter de apresentar todos os dados.

Depois de escolhidos os atributos é pedida a senha que foi usada para gerar as máscaras na aplicação inicial. São então geradas novamente as máscaras para os atributos que o owner pretende manter visíveis.

Listing 4.1: Gerar máscaras

```
1 "attributes": [  
2     {  
3         "label": atributo["label"],  
4         "value": atributo["value"], # valor original do atributo  
5         "mask": encode_base64(gerar_mascara(atributo["label"], senha))  
6     }  
7     for atributo in dcc_final["attributes"]  
8     if atributo["label"] in atributos_visiveis  
9 ],
```

Os commitments de todos os atributos são adicionados a uma lista e colocados no JSON do min_dcc em conjuntos com os atributos visíveis e as máscaras destes atributos.

Listing 4.2: Adicionar commitments

```
1 "commitments": [  
2     attr['commitment']  
3     for attr in dcc_final['attributes']  
4 ],
```

De seguida estes dados são assinados utilizando a chave privada do cartão de cidadão então o cartão de cidadão físico ainda é necessário neste passo. É utilizado RSA com o algoritmo SHA512.

Listing 4.3: Assinar dados

```
1 # Gerar a assinatura do DCC  
2 dcc_filtrado_str = json.dumps(dcc_filtrado, sort_keys=True)  
3 print("Dados a assinar: ", dcc_filtrado_str)  
4 # assinatura = assinar_dcc(chave_privada_owner, dcc_filtrado_str)  
5 assinatura, mecanismo = assinar_com_chave_privada(dcc_filtrado_str)  
6  
7 # Adicionar a assinatura ao DCC  
8 timestamp = time.time()  
9 dcc_filtrado["Owner-signature"] = {  
10     #bytes to string  
11     "signature_value": assinatura.hex(),
```



```

12     "timestamp": timestamp,
13     "crypto_system": str(mecanismo)
14 }

```

Listing 4.4: Ler chave privada e assinar

```

1 def assinar_com_chave_privada(dados):
2     try:
3         # Caminho da biblioteca PKCS#11
4         lib = '/usr/local/lib/libpteidpkcs11.so'
5         pkcs11 = PyKCS11Lib()
6         pkcs11.load(lib)
7         slots = pkcs11.getSlotList()
8         if not slots:
9             print("Nenhum cartao encontrado.")
10            exit()
11
12            # Selecionar o primeiro slot
13            slot = slots[0]
14            session = pkcs11.openSession(slot, PyKCS11.CKF_SERIAL_SESSION | PyKCS11.
15                CKF_RW_SESSION)
16
17            # Localizar a chave privada no cartao
18            priv_key_obj = None
19
20            try:
21                priv_key_obj = session.findObjects([(PyKCS11.CKA_CLASS, PyKCS11.
22                    CKO_PRIVATE_KEY), (PyKCS11.CKA_LABEL, "CITIZEN AUTHENTICATION KEY")
23                ])[0]
24                mechanism = PyKCS11.Mechanism(PyKCS11.CKM_SHA512_RSA_PKCS)
25                assinatura = session.sign(priv_key_obj, dados, mechanism)
26                assinatura_bytes = bytes(assinatura) # Converter para bytes
27                session.closeSession()
28                print("Assinatura gerada com sucesso!")
29                return assinatura_bytes, "CKM_SHA512_RSA_PKCS"

```

O min_dcc vai ter então uma assinatura do owner em cima de todos os dados com a chave do seu cartão. O min_dcc final vai ser guardado com o seguinte formato:

Listing 4.5: Min DCC

```

1 {
2     "commitments":[
3         "9bfff8165f1abf93c99bec5dd17bcda5f9f8bb6e3cc8d6d05577a9b1a2279f9d9",
4         "eb8efaed4cc544df68730bae65293ca74e8ba448cdd9b893fd94b96fb819e7a4",
5         "114b6d2001154bb8268584ed54d21bf394465285302117c003a78709a4135dd5",
6         "fee113691dcce3a8ee4a6f2368acc0c8a1d93dfea98abc845053d7cdd2dc137c"],
7     "digest_function": {
8         "type": "SHA-256"
9     },
10    "attributes": [
11        {
12            "label": "nome",
13            "value": ("Nome", "Mask")
14        },
15        {
16            "label": "morada",
17            "value": ("Rua Romana", "Mask")
18        }
19    ]
20    "chave_publica_owner": [
21        {

```

```

22         "value":"-----BEGIN PUBLIC KEY-----\n
           nMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAs8Bj62Wm2oPErsnH21zi\n
           nz4iVZ1Sj5FyoT7bf8NhmbvOg52fTFYgks3/oXtYn5\n-----END PUBLIC KEY\n
           -----\\n"
23         "description of key nature": "some description of the nature of the\n
           asymmetric cryptosystem of this key"
24     }
25 ],
26     "Issuer_signature": [
27         {
28             "value": "signature",
29             "timestamp": timestamp,
30             "description": " description of the asymmetric cryptosystem",
31             "issuer_certificate": "issuer_certificate"
32         }
33     ],
34     "Owner_signature": [
35         {
36             "value": "signature",
37             "timestamp": timestamp,
38             "description": " description of the asymmetric cryptosystem"
39         }
40     ]
41 }

```

Foi então gerado o Cartão de Cidadão digital com apenas os dados necessários.

Capítulo 5

check_dcc

5.0.1 Verificar integridade do DCC

Nesta aplicação é pedido ao utilizador um JSON de min_dcc para ser lido e verificado. É dado então ao utilizador a opção de mostrar os dados visíveis presentes no DCC.

```
Menu:
1. Ver Dados do DCC
2. Validar Integridade dos Dados
3. Sair
Escolha uma opção (1/2/3): 2
Assinatura do Issuer válida: os dados estão íntegros.
Assinatura do Owner válida: os dados estão íntegros.
Verificando compromisso do atributo 'nome', valor: 'DIOGO FILIPE ROSÁRIO DE ALMEIDA,' máscara: 'eFBPsKciKPTD0QodRpteg93tQypKjl3ArzrFiyEwv8='
O compromisso do atributo 'nome' é válido.
Verificando compromisso do atributo 'data_nascimento', valor: '2003-02-05 12:00:00+00:00,' máscara: 'HyatIHZgcH37nCUCYywcEEV3Q+7IzHigtvsTGNQzjK0='
O compromisso do atributo 'data_nascimento' é válido.
```

Figura 5.1: Ver dados do min_dcc

O utilizador tem também a opção de verificar a integridade dos dados. Quando o utilizador seleciona esta opção, primeiro é lido certificado da entidade superior (issuer) que assinou os dados. É verificado utilizando os commitments e a chave pública do owner presentes no ficheiro se a assinatura está correto e se os dados estão íntegros.

Listing 5.1: Verificar assinatura do issuer

```
1 try:
2     # Extrair dados da assinatura
3     assinatura_data = resposta['Issuer_signature'][0]
4     assinatura = bytes.fromhex(assinatura_data['value'])
5     certificado_pem = assinatura_data['issuer_certificate']
6
7     # Carregar o certificado
8     certificado = load_pem_x509_certificate(certificado_pem.encode('utf-8'),
9                                           default_backend())
9     chave_publica = certificado.public_key()
10    # Concatenar os valores dos commitments e a chave publica do owner
11    compromissos_e_chave_publica = []
12    for atributo in resposta["commitments"]:
13        compromissos_e_chave_publica.append(atributo)
14
15    compromissos_e_chave_publica.append(resposta["chave_publica_owner"][0]["
16        value"])
17
18    # Gerar os dados para assinar
19    dados_para_assinar = json.dumps(compromissos_e_chave_publica, sort_keys=True
20    )
21
22    hash_dados = hashes.Hash(hashes.SHA1(), backend=default_backend())
23    hash_dados.update(dados_para_assinar.encode())
24    digest = hash_dados.finalize()
25
26    # Verificar a assinatura
27    chave_publica.verify(
28        assinatura,
29        digest,
30        ec.ECDSA(hashes.SHA1()))
```

```

29     )
30
31     print("Assinatura do Issuer valida: os dados estao integros.")
32     return True

```

Depois é verificada a assinatura do Owner para garantir a integridade dos dados do cartão.

Listing 5.2: Verificar assinatura do owner

```

1 def validar_assinatura_owner(chave_publica, dados_para_validar, assinatura):
2     try:
3         chave_publica = serialization.load_pem_public_key(
4             chave_publica.encode('utf-8'),
5             backend=default_backend()
6         )
7
8         # Converter a assinatura de hexadecimal para bytes
9         assinatura_bytes = bytes.fromhex(assinatura)
10
11         dados_bytes = dados_para_validar.encode('utf-8')
12
13         # Verificar a assinatura utilizando a chave publica
14         chave_publica.verify(
15             assinatura_bytes, # A assinatura a ser validada
16             dados_bytes, # O hash dos dados originais
17             padding.PKCS1v15(), # O esquema de padding
18             hashes.SHA512() # O algoritmo de hash utilizado
19         )
20
21         print("Assinatura do Owner valida: os dados estao integros.")
22
23     except ValueError as e:
24         print(f"Erro ao verificar a assinatura: {e}")
25     except Exception as e:
26         print(f"Assinatura do Owner invalida ou documento adulterado: {e}")

```

Como são enviadas as máscaras dos atributos visíveis é possível calcular o commitment value desse atributo. Então depois de serem verificadas as assinaturas do owner e do issuer são então calculados os commitments values e é verificado se eles estão presentes na lista de commitments para garantir que os valores não foram alterados.

Listing 5.3: Verificar commitments

```

1 def verificarCompromisso(nome_atributo, valor_atributo, mascara, lista_compromissos)
2     :
3     # Calcular o compromisso
4     compromisso_calculado = calcular_compromisso(nome_atributo, valor_atributo,
5         mascara)
6
7     # Verificar se o compromisso calculado e igual ao compromisso fornecido
8     if compromisso_calculado in lista_compromissos:
9         print(f"O compromisso do atributo '{nome_atributo}' e valido.")
10    else:
11        print(f"O compromisso do atributo '{nome_atributo}' e invalido.")

```

Listing 5.4: Calcular commitments

```

1 # Funcao para calcular o compromisso
2 def calcular_compromisso(nome_atributo, valor_atributo, mascara):

```

```
3     compromisso = hashlib.sha256()
4     compromisso.update(nome_atributo.encode()) # Adiciona o nome do atributo
5     compromisso.update(valor_atributo.encode()) # Adiciona o valor do atributo
6     compromisso.update(mascara)
7     return compromisso.hexdigest()
```

É então apresentado ao utilizador os resultados:

```
Menu:
1. Ver Dados do DCC
2. Validar Integridade dos Dados
3. Sair
Escolha uma opção (1/2/3): 2
Assinatura do Issuer válida: os dados estão íntegros.
Assinatura do Owner válida: os dados estão íntegros.
Verificando compromisso do atributo 'nome', valor: 'DIOGO FILIPE ROSÁRIO DE ALMEIDA,' máscara: 'eFBPsKcIKPTD0QodRpteg93tQypKj13ArzrF1ycEmv8='
0 compromisso do atributo 'nome' é válido.
Verificando compromisso do atributo 'data_nascimento', valor: '2003-02-05 12:00:00+00:00,' máscara: 'HyatIHZgcH37nCUCYywcEEV3Q+7IzHigtvsTGNQzjK0='
0 compromisso do atributo 'data_nascimento' é válido.
```

Figura 5.2: Resultados do check_dcc

Capítulo 6

Algoritmos e sistemas criptográficos

Forma usados os seguintes algoritmos e sistemas criptográficos:

- Digest Function: SHA256
- Criptosystem owner key pair e signature algorithm: RSA com SHA512
- Criptosystem issuer key pair e signature algorithm: Elliptic Curve Digital Signature Algorithm (ECDSA) com SHA1