Zeutro LLC

Encryption & Data Security

OpenABE Command-line Utilities

Documentation

OpenABE Command-Line Utilities Documentation

Version 1.0

Contents

1 Introduction

The OpenABE toolkit is a C++ library that primarily implements different flavors of attribute-based encryption (ABE) algorithms that can be used across a variety of data-at-rest applications.

The OpenABE command-line utilities provide a set of programs that use the general OpenABE API for implementing ciphertext-policy attribute-based encryption (CP-ABE), key-policy attribute-based encryption (KP-ABE), and public-key encryption (PKEnc).

The OpenABE incorporates best practices in encryption algorithm design which includes security against chosen ciphertext attacks and performing authenticated encryption of large data objects. Thus, note that the programs (that comprise the OpenABE command-line utilities) use the chosen-ciphertext secure version of each encryption algorithm.

2 OpenABE CLI Description

The OpenABE command-line utilities consists of four command-line tools for performing various operations on each type of encryption algorithm supported in the OpenABE:

- `oabe_setup`: Generates master public parameters and a master secret key for any one of ABE algorithms supported.

```
OpenABE command-line: system setup utility, v1.0
usage: [ -s scheme ] [ -p prefix ] -v
-v : turn on verbose mode
-s : scheme types are 'CP' or 'KP'
-p : prefix for generated authority public \
             and secret parameter files (optional)
```

- `oabe_keygen`: Generates a private key with a given set of attributes if CP-ABE or a given policy formula if KP-ABE. This program also generates a public and private keypair for PKEnc given a user identity string.

```
OpenABE command-line: key generation utility, v1.0
usage: [ -s scheme ] [ -p prefix ] [ -i input ] [ -o output file ] -v
-v : turn on verbosity
-s : scheme types are 'PK', 'CP' or 'KP'
-i : key id for 'PK', attribute list for 'CP' and policy for 'KP'
-o : output file for generated secret key
-p : prefix for generated authority public \
             and secret parameter files (optional)
```

- `oabe_enc`: Encrypts a file according to a policy if CP-ABE, or a set of attributes if KP-ABE. Or it can encrypt a file to a specific user identity if public-key encryption.

```
OpenABE command-line: encryption utility, v1.0
usage: [ -s scheme ] [ -p prefix ] [ -e enc input ] \
               [ -i input ] [ -o output ] -v
-v : turn on verbosity
-s : scheme types are 'PK', 'CP' or 'KP'
-e : sender key Id for 'PK', policy for 'CP' \
                    or attribute list for 'KP'
-r : recipient key Id for 'PK'
-i : input file
-o : output file for ciphertext
-p : prefix for generated authority public \
             and secret parameter files (optional)
```

- `oabe_dec`: Decrypts a file that contains ciphertext using a specified private key.

```
OpenABE command-line: decryption utility, v1.0
usage: [ -s scheme ] [ -p prefix ] [ -k key ] \
               [ -i ciphertext ] [ -o output ] -v
-v : turn on verbosity
-s : scheme types are 'CP' or 'KP'
-k : recipient key Id for 'PK' and secret key file for 'CP'/'KP'
-e : sender key Id for 'PK'
-i : ciphertext file
```

```
-o : output file for plaintext
-p : prefix for generated authority public
             and secret parameter files (optional)
```

By default, the generated master public and secret parameters are stored in and loaded from the current working directory.

3 Quick Tutorial

This section provides a tutorial on how to use each program in the OpenABE command-line utilities for encrypting and decrypting files with attribute-based encryption or public-key encryption:

3.1 Ciphertext-Policy Attribute-based Encryption (CP-ABE)

In a CP-ABE system, attributes are associated with users and policies are associated with ciphertexts. A user can decrypt a ciphertext if and only if her attributes satisfy the policy.
For example, Alice has attributes `Female`, `Nurse`, `Floor=3`, `Respiratory-Specialist`. Charlie has attributes `Male`, `Doctor`, `Floor=5`, `Cardiologist`. Bob encrypts a patient's medical file for staff members matching the policy `(Doctor OR Nurse) AND (Floor <= 3 AND Floor < 5)`. Alice is able to open this patient's file but Charlie cannot.

This system provides role-based access control and works well when the encryption policies are known beforehand, *e.g.*, technical reports, HR documents, medical records. See the example below:

```
# Generate system parameters with file name prefix "org1"
./oabe_setup -s CP -p org1

# Generate key for alice
./oabe_keygen -s CP -p org1 -i "Female|Nurse|Floor=3|RespSpecialist" \
             -o aliceCP

# Generate key for charlie
```

```
./oabe_keygen -s CP -p org1 -i "Male|Doctor|Floor=5|Cardiologist" \
             -o charlieCP

# Encrypt such that alice can decrypt but charlie cannot
# Floor range portion is thesame as (Floor > 2 and Floor < 5)
./oabe_enc -s CP -p org1 -e "((Doctor or Nurse) and (Floor in (2-5)))" \
              -i input.txt -o input.cpabe

# Decrypt using charlie's key -- should fail
./oabe_dec -s CP -p org1 -k charlieCP.key -i input.cpabe \
          -o plainFail.input.txt

# Decrypt using alice's key -- should pass
./oabe_dec -s CP -p org1 -k aliceCP.key -i input.cpabe \
          -o plainOK.input.txt
```

3.2 Key-Policy Attribute-based Encryption (KP-ABE)

In a KP-ABE system, policies are associated with users (that is, their private keys) and attributes are associated with ciphertexts. A user can decrypt a ciphertext if and only if its attributes satisfy her (private key's) policy.

For example, Company Y maintains a record of its emails, where each email is encrypted with a set of attributes associated with the meta-data of that email, e.g., to, from, subject, date, IP address, etc. Later, it is discovered that sensitive information was leaked by employee Alice during the last week of February 2017 or first two weeks of March 2017. Company Y can generate a key that allows an auditor to open any emails matching the policy `(From:Alice AND (Dated = March 1-14, 2017 OR Dated = February 22-28, 2017))`. This grants an auditor access to the slice of information he needs without allowing him to view the entire email database.

This system is well-suited for granting access to slices of data (also called content-based access control). For example, releasing information to an analyst or auditor, responding to a subpoena, searching in email, cloud or other big data applications. See the example below:

```
# Generate system parameters with file name prefix "org2"
```

```
./oabe_setup -s KP -p org2

# Generate key slice for analyst
./oabe_keygen -s KP -p org2 -i "(From:Alice and (Date = March 1-14, 2017 \
                    or Date = February 22-28, 2017))" -o analystKP


# Encrypt with metadata from email
./oabe_enc -s KP -p org2 -e "From:Alice|To:Bob|Date = March 7, 2017" \
          -i input.txt -o input1.kpabe

./oabe_enc -s KP -p org2 -e "From:Alice|To:Charlie|Date = March 15, 2017" \
          -i input.txt -o input2.kpabe

# Decrypt input1.kpabe using analyst key -- should pass
./oabe_dec -s KP -p org2 -k analystKP.key \
          -i input1.kpabe -o ok.input.txt

# Decrypt input2.kpabe using analyst key -- should fail
./oabe_dec -s KP -p org2 -k analystKP.key \
          -i input2.kpabe -o fail.input.txt
```

3.3 Public-key Encryption (PKE)

For cases where traditional public-key encryption is desired, we provide such an encryption scheme in the OpenABE command-line utilities. This works in the usual way by allowing encryption with individual public keys. See the example below:

```
# Generate public and private key for alice (-i denotes the user's identifier)
./oabe_keygen -s PK -i alice

# Generate public and private key for bob
./oabe_keygen -s PK -i bob

# Generate public and private key for eve
./oabe_keygen -s PK -i eve
```

```
# Encrypt input.txt from alice to bob
./oabe_enc -s PK -e alice -r bob -i input.txt -o input.pkenc

# Note that -e = sender and -r = receiver
# Decrypt as eve -- should fail
./oabe_dec -s PK -e alice -r eve -i input.pkenc -o fail.input.txt

# Decrypt as bob -- should pass
./oabe_dec -s PK -e alice -r bob -i input.pkenc -o ok.input.txt
```

4 Attributes and Data Structures

4.1 Attributes

Attributes can be any alphanumeric string and are not restricted or fixed at system initialization time. Attributes also includes printable symbols/characters such as `$, &, #, %` and so on. We illustrate here our specific approach for representing attributes. In the OpenABE, attributes are in two parts: `{type}: {value}`. An optional $type$ part that indicates the type of attribute being represented and a $value$ part that represents the actual attribute string. This $type$ field serves as a way of capturing namespaces or domains for attributes. For example, `system:attribute1` could indicate that a specific $attribute1$ is a system-only type attribute. Alternatively, the $type$ could be used to encode an authority's identity and domain of attributes (e.g. for Multi-Authority ABE).

We also represent numbers as attributes. Specifically, to represent the numerical attribute `a = k` for some n-bit integer k, we convert the comparison into a $binary$ representation which produces n (non-numerical) attributes which specify the value of each bit in k (up to 32-bits). As an optimization, we support specifying how many bits to use when representing the value k with a simple encoding such as `a = k#b`. Here, b denotes the number of bits necessary to represent k. Note that we do a sanity check to make sure that all of k can be represented with b bits.

4.2 Keys and Ciphertexts

Each OpenABE key and ciphertext structure includes a header that describes some metadata. This metadata includes the encryption scheme identifier, the underlying elliptic curve (or pairing curve) identifier, the OpenABE library version number and a unique 128-bit object identifier. In the case of the ciphertext, the header metadata is protected against tampering attacks along with the ciphertext body.

5 Benchmarking

The OpenABE is built on top of the abstract Zeutro Math library which supplies all of our elliptic-curve operations. We instantiate our schemes using the state-of-the-art Barreto-Naehrig (BN) curves with the embedding degree `k = 12` (or commonly referred to as `BN-254`). This particular asymmetric curve is known to yield a very efficient pairing implementation and a security level equivalent to `AES-128`. As a result, this boosts the overall performance of ABE scheme implementations over prior efforts. Other benefits of BN curves include the ability to compress the representation of group elements. This directly translates to making ABE ciphertexts more compact which considerably reduces transmission costs.

We include a benchmark utility for all the ABE schemes provided in the OpenABE:

```
Math Library: RELIC
OpenABE benchmark utility, v1.0
Usage bench_libopenabe: [ scheme => 'CP' or 'KP' ] [ iterations ] \
               [ attributes ] [ 'fixed' or 'range' ] [ 'cpa' or 'cca']
-scheme: the type of ABE scheme to benchmark
-iterations: the number of iterations per test
-attributes: the number of attributes in the policy or \
                 attribute list for encryption
-'fixed' or 'range': run with a fixed number of attributes \
                     or as a range from 1 to num. attributes
-'cpa' or 'cca': chosen-plaintext secure vs chosen-ciphertext \
                 secure versions
```

For example, the command below shows how to benchmark the CCA-secure KP-ABE implementation with 100 attributes for encryption (averaged over 10 iterations). Moreover, the generated decryption key policy will have 100 attributes and each attribute will be involved in the decryption.

```
bench_libopenabe KP 10 100 fixed cca
```