libfenc: The Functional Encryption Library (v0.1a)

Reference Manual

Matthew Green

March 13, 2017

Contents

1 Introduction **5**
1.1 Background: Functional Encryption 5
1.2 Overview of the Functional Encryption Library 6
1.2.1 Supported encryption schemes 6
1.2.2 What libfenc doesn't do 6
1.2.3 Dependencies 6
1.3 Outline of this Manual 6

2 Using the Library **7**
2.1 Building the Library 7
2.2 Using libfenc in an application 7
2.2.1 Compiling the application 7
2.2.2 A brief tutorial 7

3 The API **9**

A Scheme Details **13**
A.1 Lewko-Sahai-Waters KP-ABE [LSW08] 13

B License and Attribution **15**

Chapter 1

Introduction

The Functional Encryption Library (libfenc) is a general-purpose framework for implementing *functional encryption* schemes. Functional encryption is a generalization that encompasses a number of novel encryption technologies, including Attribute-Based Encryption (ABE), Identity-Based Encryption (IBE) and several other new primitives.

1.1 Background: Functional Encryption

In a functional encryption scheme, encryptors associate ciphertext values with some value X. Decryptors may request a decryption key associated with a value Y; these are produced by a trusted authority known as the Private Key Generator (PKG). For some function $F : a \times b \rightarrow \{0, 1\}$ associated with the encryption scheme, cecryption is permitted if and only if the following relationship is satisfied:

$$F(X, Y) = 1$$

The choice of encryption scheme typically defines the function F as well as the form of the inputs X, Y. This formulation encompasses the following encryption types:

1. *Identity Based Encryption.* In an IBE scheme [Sha84, BF01], both X and Y are identities (arbitrary bitstrings $\{0, 1\}^*$) and F outputs 1 iff $X = Y$.

2. *Key-Policy Attribute Based Encryption.* In a KP-ABE [**?**] scheme, the value X is a list of attributes associated with the ciphertext, while the key-assocated value Y contains a complex "policy", which is typically represented by an access tree. The function F outputs 1 iff the ciphertext's attributes satisfy the policy.

3. *Ciphertext-Policy Attribute Based Encryption.* In a CP-ABE [] scheme, the value X is an attribute policy that will be associated with the ciphertext, while Y is an attribute list associated with the key.

1.2 Overview of the Functional Encryption Library

1.2.1 Supported encryption schemes

The Functional Encryption Library currently implements the following encryption schemes. This list is expected to grow in later releases.

1. **Lewko-Sahai-Waters KP-ABE [LSW08].** An efficient ABE scheme supporting non-monotonic access structures. Secure under the q-MEBDH assumption in prime-order bilinear groups.

1.2.2 What libfenc doesn't do

1.2.3 Dependencies

1.3 Outline of this Manual

The remainder of this manual is broken into several sections. Chapter **??** gives a basic overview of the library's usage, from the point of view of an application writer. Chapter 3 provides a canonical description of the library API. Finally, Chapter **??** provides an detailed description of the schemes that are currently supported by libfenc, with some details on their internal workings.

Chapter 2

Using the Library

This section provides a brief tutorial on libfenc, tailored for the developers who wish to use the library in their applications. We first describe the process of building and installing the library, then give some examples of how the library is used in practice. For a full description of the library API, see chapter 3.

2.1 Building the Library

2.2 Using libfenc in an application

2.2.1 Compiling the application

The build process above produces the static library `libfenc.la` which should be located in a known location in your system.

2.2.2 A brief tutorial

The basic unit of the Functional Encryption Library is the *encryption context*. This is an abstract data structure responsible for storing the scheme type as well as the public and/or secret parameters associated with the scheme. An application may instantiate multiple encryption contexts if desired, running the same or different encryption schemes.

Most API routines return an error code of type `FENC_ERROR`. Always be sure to check that the returned value is FENC_ERROR_NONE, or the library may not operate correctly. Error codes can be converted into strings using the `libfenc_error_to_string()` call.

1. Initialize the libfenc library. An application must execute this routine before conducting any operations with the library:

   ```
   err_code = libfenc_init();
   ```

2. Next, create an encryption context for a given scheme type. The caller is responsible for allocating the fenc_context structure which is passed to this routine. A list of encryption schemes is provided in §**??**:

   ```
   fenc_context context;
   err_code = libfenc_create_context(&context, FENC_SCHEME_LSW);
   ```

3. The next step is to provision the scheme with a set of parameters. For most schemes, only public parameters are needed for encryption. Secret parameters will also be needed if the application wishes to extract decryption keys.

   Keys may be loaded from an external source, or they can be generated from scratch. To generate both the public and secret parameters, use the `libfenc_gen_params` call as in the following snippet:

   ```
   fenc_global_params global_params;
   err_code = libfenc_gen_params(&context, &global_params);
   ```

Library Initialization.

Chapter 3

The API

This chapter describes the libfenc API, which includes the data structures and function calls necessary to write compatible applications. For a brief description of the calling sequence, see the tutorial of §2.2.2.

Bibliography

[BF01] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the Weil Pairing. In *CRYPTO '01*, volume 2139 of LNCS, pages 213–229, 2001.

[LSW08] Allison Lewko, Amit Sahai, and Brent Waters. Revocation systems with very small private keys. Cryptology ePrint Archive, Report 2008/309, 2008. `http://eprint.iacr.org/2008/309`.

[Sha84] Adi Shamir. Identity-based cryptosystems and signature schemes. In *CRYPTO '84*, volume 196 of LNCS, pages 47–53, 1984.

Appendix A

Scheme Details

A.1 Lewko-Sahai-Waters KP-ABE [LSW08]

This section documents some of the implementation-specific decisions made in implementing the Lewko-Sahai-Waters scheme. In particular, the original scheme description specifies the use of symmetric groups; in our implementation we are required to specify which elements belong to $\mathbb{G}_1$ and $\mathbb{G}_2$.

The following quotes the original scheme description from [LSW08] with minor modifications. All group elements marked normally (*e.g.,* g) are in $\mathbb{G}_1$, while those marked with a bar ($\bar{g}$) are in $\mathbb{G}_2$.

Setup. The setup algorithm chooses generators $g \in \mathbb{G}_1, \bar{g} \in \mathbb{G}_2$ and picks random exponents $z, \alpha', \alpha'', b \in \mathbb{Z}_p$. We define $h = g^z \in \mathbb{G}_1, \bar{h} = \bar{g}^z \in \mathbb{G}_2, \alpha = \alpha' \cdot \alpha''$, $g_1 = g^{\alpha'}$ and $g_2 = g^{\alpha''}$. The public parameters are published as the following, where H is a random oracle that outputs elements of the elliptic curve group:

$$params = (g, g^b, g^{b^2}, h^b, e(g,g)^\alpha, H(\cdot)).$$

The authority keeps (α', α'', b) as the master key MK.

Encryption $(M, \gamma, params)$. To encrypt a message $M \in \mathbb{G}_T$ under a set of d attributes $\gamma \subset \mathbb{Z}_p^*$, choose a random value $s \in \mathbb{Z}_p$, and choose a random set of d values $\{s_x\}_{x\in\gamma}$ such that $s = \sum_{x\in\gamma} s_x$. Output the ciphertext as

$$E = (\gamma, E^{(1)} = Me(g,g)^{\alpha\cdot s}, E^{(2)} = g^s, \{E_x^{(3)} = H(x)^s\}_{x\in\gamma}, \\ \{E_x^{(4)} = g^{b\cdot s_x}\}_{x\in\gamma}, \{E_x^{(5)} = g^{b^2\cdot s_x x} h^{b\cdot s_x}\}_{x\in\gamma})$$

Key Generation $(\tilde{\mathbb{A}}, \mathrm{MK}, params)$. This algorithm outputs a key that enables the user to decrypt an encrypted message *only* if the attributes of that ciphertext satisfy the access structure $\tilde{\mathbb{A}}$. We require that the access structure $\tilde{\mathbb{A}}$ is $NM(\mathbb{A})$ for some monotonic access structure $\mathbb{A}$, (see [**?**] for a definition of the $NM(\cdot)$ operator) over a set $\mathcal{P}$ of attributes, associated with a linear secret-sharing scheme Π. First, we apply the linear secret-sharing mechanism Π to obtain shares $\{\lambda_i\}$ of the secret α'. We denote the party corresponding to the share λ_i as $\breve{x}_i \in \mathcal{P}$, where x_i is

the attribute underlying $\breve{x}_i$. Note that $\breve{x}_i$ can be primed (negated) or unprimed (non negated). For each i, we also choose a random value $r_i \in \mathbb{Z}_p$.

The private key D will consist of the following group elements: For every i such that $\breve{x}_i$ is *not* primed (i.e., is a non-negated attribute), we have

$$D_i = (D_i^{(1)} = g_2^{\lambda_i} \cdot H(x_i)^{r_i}, D_i^{(2)} = g^{r_i})$$

For every i such that $\breve{x}_i$ is primed (i.e., is a negated attribute), we have

$$D_i = (D_i^{(3)} = g_2^{\lambda_i} g^{b^2 r_i}, D_i^{(4)} = g^{r_i b x_i} h^{r_i}, D_i^{(5)} = g^{-r_i})$$

The key D consists of D_i for all shares i.

Decryption (E, D). Given a ciphertext E and a decryption key D, the following procedure is executed: (All notation here is taken from the above descriptions of E and D, unless the notation is introduced below.) First, the key holder checks if $\gamma \in \tilde{\mathbb{A}}$ (we assume that this can be checked efficiently). If not, the output is $\perp$. If $\gamma \in \tilde{\mathbb{A}}$, then we recall that $\tilde{\mathbb{A}} = NM(\mathbb{A})$, where $\mathbb{A}$ is an access structure, over a set of parties $\mathcal{P}$, for a linear secret sharing-scheme Π. Denote $\gamma' = N(\gamma) \in \mathbb{A}$, and let $I = \{i : \breve{x}_i \in \gamma'\}$. Since γ' is authorized, an efficient procedure associated with the linear secret-sharing scheme yields a set of coefficients $\Omega = \{\omega_i\}_{i \in I}$ such that $\sum_{i \in I} \omega_i \lambda_i = \alpha$. (Note, however, that these λ_i are not known to the decryption procedure, so neither is α.)

For every positive (non negated) attribute $\breve{x}_i \in \gamma'$ (so $x_i \in \gamma$), the decryption procedure computes the following:

$$\begin{aligned} Z_i &= e\left(D_i^{(1)}, E^{(2)}\right) / e\left(D_i^{(2)}, E_i^{(3)}\right) \\ &= e\left(g_2^{\lambda_i} \cdot H(x_i)^{r_i}, g^s\right) / e\left(g^{r_i}, H(x)^s\right) \\ &= e\left(g, g_2\right)^{s\lambda_i} \end{aligned}$$

For every negated attribute $\breve{x}_i \in \gamma'$ (so $x_i \notin \gamma$), the decryption procedure computes the following, following a simple analogy to the basic revocation scheme:

$$\begin{aligned} Z_i &= \frac{e\left(D_i^{(3)}, E^{(2)}\right)}{e\left(D_i^{(4)}, \prod_{x \in \gamma} \left(E_x^{(4)}\right)^{1/(x_i - x)}\right) \cdot e\left(D_i^{(5)}, \prod_{x \in \gamma} \left(E_x^{(5)}\right)^{1/(x_i - x)}\right)} \\ &= e\left(g, g_2\right)^{s\lambda_i} \end{aligned}$$

Finally, the decryption is obtained by computing

$$\frac{E^{(1)}}{\prod_{i \in I} Z_i^{\omega_i}} = \frac{M e(g,g)^{s\alpha}}{e(g, g_2)^{s\alpha'}} = M$$

Note on Efficiency and Use of Random Oracle Model.

Appendix B

License and Attribution

The Functional Encryption Library is distributed under the Internal Distribution Only license. For complete details of the license terms, see http://nowhere.nowhere. Please note that the library fundamentally depends on the Stanford Pairing Based Crypto Library (PBC) and the GNU Multiprecision Library. Developers must obtain these libraries and abide by their license conditions.

Additionally, the library incorporates source code from the following projects:

1. The SHA1 implementation is due to Paul E. Jones (paulej@packetizer.com) and is drawn from RFC 1374 under the following terms:

   This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included on all such copies and derivative works. However, this document itself may not be modified in any way, such as by removing the copyright notice or references to the Internet Society or other Internet organizations, except as needed for the purpose of developing Internet standards in which case the procedures for copyrights defined in the Internet Standards process must be followed, or as required to translate it into languages other than English.

2. Some versions of this library may incorporate code from the OpenSSL project, which is distributed under the following license conditions:

```
/* ====================================================================
 * Copyright (c) 1998-2008 The OpenSSL Project.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright
```

```
*    notice, this list of conditions and the following disclaimer in
*    the documentation and/or other materials provided with the
*    distribution.
*
* 3. All advertising materials mentioning features or use of this
*    software must display the following acknowledgment:
*    "This product includes software developed by the OpenSSL Project
*    for use in the OpenSSL Toolkit. (http://www.openssl.org/)"
*
* 4. The names "OpenSSL Toolkit" and "OpenSSL Project" must not be used to
*    endorse or promote products derived from this software without
*    prior written permission. For written permission, please contact
*    openssl-core@openssl.org.
*
* 5. Products derived from this software may not be called "OpenSSL"
*    nor may "OpenSSL" appear in their names without prior written
*    permission of the OpenSSL Project.
*
* 6. Redistributions of any form whatsoever must retain the following
*    acknowledgment:
*    "This product includes software developed by the OpenSSL Project
*    for use in the OpenSSL Toolkit (http://www.openssl.org/)"
*
* THIS SOFTWARE IS PROVIDED BY THE OpenSSL PROJECT ``AS IS'' AND ANY
* EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
* IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
* PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE OpenSSL PROJECT OR
* ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
* SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
* NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
* LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
* HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
* STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
* ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
* OF THE POSSIBILITY OF SUCH DAMAGE.
* ====================================================================
*
* This product includes cryptographic software written by Eric Young
* (eay@cryptsoft.com).  This product includes software written by Tim
* Hudson (tjh@cryptsoft.com).
*
*/
```