Overview and User's Guide

The JHU-MIT Proxy Re-cryptography Library (PRL) v0.1

Matthew Green

1 Introduction

NOTE: This document is incomplete and does not include a full description of library usage.

The JHU-MIT Proxy Re-cryptography Library (PRL) is an open source encryption library that implements several proxy re-encryption schemes. The algorithms were developed by researchers at the Johns Hopkins University and the Massachusetts Institute of Technology, and is based on original research by Giuseppe Ateniese, Kevin Fu, Matthew Green and Susan Hohenberger [2, 4, 3]. The library is distributed in source code form under a BSD license, and requires the MIRACL cryptographic library [9], which is distributed separately.[1]

The long-term goal of this project is to collect a number of new proxy re-encryption and re-signature schemes into a single location so that researchers can evaluate their suitability for various applications. The current library implements two public-key proxy re-encryption schemes. In upcoming versions of this library we plan to implement additional primitives, including the proxy re-signature schemes of [5], new proxy re-encryption schemes from [7], and an Identity-Based re-encryption scheme from [6].

Important Disclaimer: The PRL was developed for research purposes, and is substantially untested. *Use at your own risk.*

2 Overview

The core of the PRL is a C++ implementation of the proxy re-encryption schemes described in [3]. The library is intended for use by C++ or C programs, though in principle it is possible to access library functions from higher-level languages such as Python or Perl.

Overview of Proxy Re-encryption. Proxy re-encryption is a form of public-key encryption that allows a user Alice to "delegate" her decryption rights to another user Bob. Alternative solutions to this problem typically require Alice to divulge her secret key to Bob, or to enroll in some sort of key escrow system. This may be undesirable for a variety of reasons. In a proxy re-encryption scheme, Alice delegates a semi-trusted proxy to translate ciphertexts encrypted under her key into ciphertexts encrypted under Bob's key. Once delegated, the proxy operates independently of Alice. The proxy is considered "semi-trusted" because it does not see the content of the messages being translated, nor can it re-encrypt Alice's messages to users for whom Alice has not granted decryption rights.

For example, if Alice and Bob are both users in an encrypted email system, Alice might wish to temporarily forward messages to Bob while she is away. In this setting, she might instruct the mail server

[1] MIRACL is distributed under a proprietary license, but is free for "non-commercial purposes".

(proxy) to automatically re-encrypt her incoming mail to Bob's key. Even if a malicious system administration wished to read Alice's mail, the message content would be unavailable to anyone besides Alice or Bob. When Alice returns, she can revoke Bob's decryption rights by instructing the proxy to cease re-encrypting messages to Bob.

To delegate her decryption rights to Bob, Alice generates a "delegation key" (or "re-encryption key"), and sends this key to the proxy. The proxy uses this key to translate messages from Alice's key to Bob's key. The schemes implemented by the PRL are *unidirectional*. In a unidirectional scheme, delegations are "one-way", *i.e.,* the proxy can re-encrypt Alice's messages to Bob, but cannot re-encrypt Bob's messages to anyone. Furthermore, Alice can generate a delegation key (to Bob) using only Bob's public key (and her secret key). It is not necessary that Bob be online or even know that delegation has taken place.

For more detailed information on proxy re-encryption and its applications, see [3].

Limitations of Proxy Re-encryption. Proxy re-encryption schemes have several limitations. First, collusion between a "delegatee" Bob and the proxy undermines Alice's security. Any party who obtains both a delegation key $rk_{Alice \rightarrow Bob}$ and Bob's secret key sk_{Bob}, is capable of decrypting any of Alice's ciphertexts—by first re-encrypting from Alice's key to Bob's, and then decrypting using Bob's secret key. Although in principle the colluders have not obtained Alice's actual secret key, the combination of $(rk_{Alice \rightarrow Bob} + sk_{Bob})$ is functionally the same.[2] However, note that in a unidirectional system, the delegatee Bob's security is not at risk should Alice collude with the proxy. This follows from the fact that the proxy's delegation key $rk_{Alice \rightarrow Bob}$ cannot be used to re-encrypt Bob's ciphertexts.

The schemes implemented in the PRL are *single-use*: the proxy cannot re-encrypt an already re-encrypted message. This prevents repeated re-encryption of the same ciphertext (*e.g.,* Alice→Bob, Bob→Charlie, etc.). *Multi-use* schemes do exist, but they typically are typically either *bidirectional*, or result in ciphertext expansion upon each re-encryption. Future versions of the PRL may implement multi-use schemes.

2.1 Components of a Proxy Re-encryption System

A proxy re-encryption deployment may support an arbitrary number of users. As in a standard public-key encryption scheme, each user generates a public/secret keypair (see below) and distributes the public key. A deployment may encompass zero or more re-encrypting proxies. A given proxy can be provisioned with an arbitrary number of delegation keys (for delegations such as Alice→Bob, Charlie→Dave, etc.) There is nothing "special" about a re-encryption proxy, beyond the fact that (a) it possesses the re-encryption algorithm, and (b) some user has provisioned it with a delegation key.

Proxy re-encryption schemes require the following data components:

- **Public Parameters.** All users in a proxy re-encryption deployment share a common set of public parameters. These parameters may be fixed (specified as part of the scheme), or they may differ between deployments.[3] Note that these parameters are public, and can be generated without the assistance of a trusted party. The need for globally-shared parameters is an important difference between a proxy re-encryption scheme and many other public-key encryption schemes.

- **Public and Secret Keypair.** Each user in a deployment generates a public/secret keypair. As with a standard public-key encryption scheme, the public key is given out to the world (preferable in

[2] Note that some of the schemes implemented by the PRL provide a weak protection known as "master secret security". See [3] for more details.

[3] However, users may only delegate decryption rights to other users who share the same public parameters, *e.g.,* users in the same deployment.

authenticated form), while the secret key remains known only to the user. When Bob sends a message to Alice, he encrypts using Alice's public key, and Alice decrypts using her secret key.

- **Delegation Keys.** Each user may generate an arbitrary number of delegation (re-encryption) keys, denoted herein as *e.g.,* $rk_{Alice \rightarrow Bob}$. To generate $rk_{Alice \rightarrow Bob}$, Alice combines her secret key with Bob's public key using an algorithm provided by the library. The resulting delegation key may then be transmitted (securely) to a re-encryption proxy.

- **Ciphertexts.** Ciphertexts are created when a user encrypts a message (plaintext) under some public key. The PRL defines three types of ciphertext:

  1. *First-level* ciphertexts have a structure that cannot be re-encrypted a proxy. The use of first-level ciphertexts is appropriate for certain applications where the sender wishes to ensure that *only* the specified recipient will decrypt the ciphertext.
  2. *Second-level* ciphertexts can be re-encrypted by a proxy. This is the "standard" form of ciphertext in a proxy re-encryption scheme.
  3. *Re-encrypted* ciphertexts result when a proxy re-encrypts a second-level ciphertext. In some schemes, re-encrypted ciphertexts have the same form as first-level ciphertexts. In this case, the delegatee Bob may not learn whether the ciphertext was originally encrypted to him, or to some other user.

2.2 Schemes

The PRL implements two proxy re-encryption schemes. Both make use of the Tate pairing in supersingular elliptic-curve groups. An extensive survey on pairing-based cryptography can be found in [8].

The schemes are described below:

- **PRE1.** This algorithm is defined in §3 of [3] under the heading "A Third Attempt". This algorithm is secure under the Decisional Bilinear Diffie-Hellman Assumption (DBDH).

- **PRE2.** This algorithm is defined in §3 of [3] under the heading "A Second Attempt". This algorithm is secure under the Decisional Bilinear Inversion assumption (DBDHI).

Each of these algorithms operate on short plaintexts (of sizes ranging from 192-512 bits, depending on the parameters used). In principle, these algorithms may be used to encrypt arbitrary bit strings. In practice, however, it is more appropriate to use the PRL algorithms to encrypt symmetric encryption keys (*e.g.,* AES session keys), which will in turn be used to encrypt some data payload. The PRL does not perform symmetric encryption or session key generation— such functionality must be implemented separately by the developer.[4]

3 Building and Installing the Library

The PRL is distributed in source-code form, and is targeted for Linux platforms. However, with small modifications the library should compile on other Unix-type systems (provided that the MIRACL library is

[4]MIRACL implements AES encryption and decryption. Alternatively, the OpenSSL cryptographic library [1] implements a wide variety of encryption, message authentication and signature algorithms.

supported), and on Windows platforms. The Linux build process outlined below assumes a system with `gcc` and other common utilities installed. To build the library:

1. Obtain the MIRACL library (`miracl3.zip`) from `http://www.shamus.ie/`. Unzip and build the library archive file using the following commands:

   ```
   mkdir miracl/
   unzip -j -aa -L miracl3.zip miracl/
   cd miracl/
   bash linux
   ```

   These commands unzip the contents of `miracl3.zip` into the directory `miracl/`— ignoring the directory structure specified within the zip file— and build the library. See the MIRACL documentation if this step does not succeed.

2. Untar the file `proxylib0_1.tar.gz` in the same base directory in which you created the `miracl/` directory (this is important: the PRL build scripts looks for the `miracl/` directory here). Next, build the library via the following commands:

   ```
   cd proxylib/src/
   make
   ```

3. Run `./proxylib_test` at the command line. This utility runs diagnostic tests on the library to ensure that it is correctly built and ready for use.

By default, the PRL is built as a static library file (`proxylib.a`). This library contains the proxy re-encryption routines, but does *not* contain the MIRACL code required to actually use the library. An application using the PRL must link against both `proxylib.a` and `miracl.a` (which can be found in the `miracl/` directory). Users who find this inconvenient may wish to generate a single library file containing all necessary code. To do this, run the command "`make withmiracl`". This will generate a single combined library `proxylibmiracl.a` that embeds the miracl object code.[5]

4 Using the Library

The proxy re-encryption library performs the following functions:

- Parameter Generation
- Keypair Generation
- Encryption and Decryption
- Delegation (Re-encryption) Key Generation
- Re-encryption
- Serialization/Deserialization of parameters, public keys, and ciphertexts.

Setup functions. All users in a proxy re-encryption deployment share global parameters.

Encryption functions.

- `CIPH_TYPE_FIRST_LEVEL  = 0`
- `CIPH_TYPE_SECOND_LEVEL = 1`
- `CIPH_TYPE_REENCRYPTED = 2`

[5] If you choose this option, you should not link miracl.a to your program, as the linker will object to the duplicated code.

References

[1] The OpenSSL Project. http://openssl.org.

[2] Giuseppe Ateniese, Kevin Fu, Matthew Green, and Susan Hohenberger. Improved Proxy Re-encryption Schemes with Applications to Secure Distributed Storage. In *the 12th Annual Network and Distributed System Security Symposium*, pages 29–43, 2005. Full version available at http://eprint.iacr.org/2005/028.

[3] Giuseppe Ateniese, Kevin Fu, Matthew Green, and Susan Hohenberger. Improved proxy re-encryption schemes with applications to secure distributed storage. Cryptology ePrint Archive, Report 2005/028, 2005. http://eprint.iacr.org/.

[4] Giuseppe Ateniese, Kevin Fu, Matthew Green, and Susan Hohenberger. Improved proxy re-encryption schemes with applications to secure distributed storage. *ACM Trans. Inf. Syst. Secur.*, 9(1):1–30, 2006.

[5] Giuseppe Ateniese and Susan Hohenberger. Proxy re-signatures: new definitions, algorithms, and applications. In *CCS '05: Proceedings of the 12th ACM conference on Computer and Communications security*, pages 310–319, New York, NY, USA, 2005. ACM Press.

[6] Matthew Green and Giuseppe Ateniese. Identity based proxy re-encryption. In *Applied Cryptography and Network Security (ACNS) 2007*, 2007.

[7] Susan Hohenberger, Guy Rothblum, Abhi Shelat, and Vinod Vaikuntanathan. Securely obfuscating re-encryption. In *Theory of Cryptography Conference (TCC), 2007*, 2007.

[8] Alfred J. Menezes. An introduction to pairing-based cryptography. http://www.math.uwaterloo.ca/~ajmeneze/publications/pairings.pdf.

[9] Shamus Software Ltd. Multiprecision Integer and Rational Arithmetic C/C++ Library (MIRACL). http://www.shamus.ie/.

A Credits

The JHU-MIT Proxy Re-cryptography Library was developed by Matthew Green, based on algorithms by Giuseppe Ateniese, Kevin Fu, Matthew Green and Susan Hohenberger.

B Copyright and Software License

Patented Software Release Agreement

This Agreement, effective as of March 1, 2007 is between the Massachusetts Institute of Technology (”MIT”), a non-profit institution of higher education, and you (YOU).

WHEREAS, M.I.T. has developed certain software and technology pertaining to M.I.T. Case No. 11977, ”Unidirectional Proxy Re-Encryption,” by Giuseppe Ateniese, Kevin Fu, Matt Green and Susan Hohenberger (PROGRAM); and

WHEREAS, M.I.T. is a joint owner of certain right, title and interest to a patent pertaining to the technology associated with M.I.T. Case No. 11977 ”Unidirectional Proxy Re-Encryption,” (PATENTED INVENTION”); and

WHEREAS, M.I.T. desires to aid the academic and non-commercial research community and raise awareness of the PATENTED INVENTION and thereby agrees to grant a limited copyright license to the PROGRAM for research and non-commercial purposes only, with M.I.T. retaining all ownership rights in the PATENTED INVENTION and the PROGRAM; and

WHEREAS, M.I.T. agrees to make the downloadable software and documentation, if any, available to YOU without charge for non-commercial research purposes, subject to the following terms and conditions.

THEREFORE:

1. Grant.

(a) Subject to the terms of this Agreement, M.I.T. hereby grants YOU a royalty-free, non-transferable, non-exclusive license in the United States for the Term under the copyright to use, reproduce, display, perform and modify the PROGRAM solely for non-commercial research and/or academic testing purposes.

(b) MIT hereby agrees that it will not assert its rights in the PATENTED INVENTION against YOU provided that YOU comply with the terms of this agreement.

(c) In order to obtain any further license rights, including the right to use the PROGRAM or PATENTED INVENTION for commercial purposes, YOU must enter into an appropriate license agreement with M.I.T.

2. Disclaimer. THE PROGRAM MADE AVAILABLE HEREUNDER IS ”AS IS”, WITHOUT WARRANTY OF ANY KIND EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, NOR REPRESENTATION THAT THE PROGRAM DOES NOT INFRINGE THE INTELLECTUAL PROPERTY RIGHTS OF ANY THIRD PARTY. MIT has no obligation to assist in your installation or use of the PROGRAM or to provide services or maintenance of any type with respect to the PROGRAM. The entire risk as to the quality and performance of the PROGRAM is borne by YOU. YOU acknowledge that the PROGRAM may contain errors or bugs. YOU must determine whether the PROGRAM sufficiently meets your requirements. This disclaimer of warranty constitutes an essential part of this Agreement. 3. No Consequential Damages; Indemnification. IN NO EVENT SHALL MIT BE LIABLE TO YOU FOR ANY LOST PROFITS OR OTHER INDIRECT, PUNITIVE, INCIDENTAL OR CONSEQUENTIAL DAMAGES RELATING TO THE SUBJECT MATTER OF THIS AGREEMENT.

4. Copyright. YOU agree to retain M.I.T.'s copyright notice on all copies of the PROGRAM or portions thereof.

5. Export Control. YOU agree to comply with all United States export control laws and regulations controlling the export of the PROGRAM, including, without limitation, all Export Administration Regulations of the United States Department of Commerce. Among other things, these laws and regulations prohibit, or require a license for, the export of certain types of software to specified countries.

6. Reports, Notices, License Request. Reports, any notice, or commercial license requests required or permitted under this Agreement shall be directed to:

Director Massachusetts Institute of Technology Technology Licensing Office, Rm NE25-230 Five Cambridge Center, Kendall Square Cambridge, MA 02142-1493

7. General. This Agreement shall be governed by the laws of the Commonwealth of Massachusetts. The parties acknowledge that this Agreement sets forth the entire Agreement and understanding of the parties as to the subject matter. BY CLICKING ON THE ”ACCEPT” BUTTON AT THE END OF THIS AGREEMENT, YOU ARE CONSENTING TO BE BOUND BY ALL OF THE TERMS OF THIS AGREEMENT. IF YOU DO NOT AGREE TO ALL THE TERMS OF THIS AGREEMENT, CLICK THE ”DO NOT ACCEPT” BUTTON, AND THE INSTALLATION/DOWNLOAD PROCESS WILL NOT CONTINUE.