Umbral: a threshold proxy re-encryption scheme

David Nuñez

Abstract

This document describes the Umbral proxy re-encryption scheme used by NuCypher KMS [1]. Umbral is a threshold proxy re-encryption scheme based on ECIES-KEM [2], the BBS98 proxy re-encryption scheme [3], and Shamir's Secret Sharing [4]. With Umbral, Alice (which in the generic name for data owners in NuCypher KMS) can delegate decryption rights to Bob for any ciphertext intended to her, through a re-encryption process performed by a set of N semi-trusted proxies. When at least t of these proxies (out of N) participate by performing re-encryption, Bob is able to combine these independent re-encrytions and decrypt the original message using his private key.

1 Introduction

2 Preliminaries

2.1 Notation

Although the additive notation is the norm when dealing with elliptic curve cryptography, in this document we adopt the multiplicative notation to express the operations in the elliptic curve group, which is the usual approach in the proxy re-encryption literature (where schemes are usually defined in generic groups).

2.2 Proxy Re-Encryption

(TODO: General description of proxy re-encryption, properties, etc)

2.3 ECIES

The Asymmetric Encryption Scheme defined in standard ANSI X9.63 [2], also known as Elliptic Curve Integrated Encryption Scheme (ECIES), is a hybrid encryption algorithm based on elliptic curve cryptography, symmetric encryption and message authentication codes. This algorithm is of public knowledge, and variants have been standardized also by ISO/IEC 18033-2 [5] and IEEE P1363A [6]. A comparison of the different variants of ECIES can be found in [7].

When producing a ciphertext with ECIES, the sender first creates an ephemeral public key and uses it for a Diffie-Hellman key agreement together with the public key of the intended recipient. The resulting shared secret is used to create the keys for the symmetric encryption and message authentication code algorithms used internally. The final ciphertext consists of the ephemeral public key, as it is necessary for decryption, and the output of the symmetric encryption and message authentication code.

A Sender can be any entity that generates data and wants to send it confidentially to a receiver, in the form of a ciphertext encrypted under the public key of the receiver. A Receiver can be any entity that is entitled to read ciphertexts encrypted under his public key. We distinguish two types of receivers. The original receiver, which is the recipient of the data originally intended by the sender, and the delegated receiver, whom the original receiver entrusts to be able to decrypt ciphertexts initially intended to him. Therefore, there is a relation of delegation between the original receiver and the delegated one. The Intermediary is an entity that controls the process of switching the public key of ciphertexts, from the public key of the original receiver to the public key of a delegated receiver, without being able to learn anything from the data. The Intermediary needs a key-switching key between the original and delegated receivers in order to be able to perform the key-switching process.

3 The Umbral PRE cryptosystem

In this section we present the Umbral PRE cryptosystem, defined by the following algorithms:

- $\mathsf{KeyGen}()$: Sample $x \in \mathbb{Z}_q$ uniformly at random, compute g^x and output the keypair $(pk, sk) = (g^x, x)$.
- $\mathsf{ReKeyGen}(sk_A, sk_B, N, t)$: On input the secret keys $sk_A = a$ and $sk_B = b$, a number of fragments N, and a threshold t, the re-encryption key generation algorithm $\mathsf{ReKeyGen}$ computes N fragments of the re-encryption key between A and B. First, it randomly samples $t-1$ elements $f_i \in \mathbb{Z}_q$, with $1 \leq i \leq t-1$, and computes $f_0 = a \cdot b^{-1} \bmod q$. The next step is using these values, including f_0, to construct a polynomial $f(x) \in \mathbb{Z}_q[x]$ of degree $t-1$, such that $f(x) = f_0 + f_1 x + f_2 x^2 + ... + f_{t-1} x^{t-1}$. Next, it randomly samples a set $ID = \{id_j \in \mathbb{Z}_q \mid 1 \leq j \leq N\}$. The algorithm outputs the set of re-encryption key fragments $KF = \{(id_j, f(id_j)) \mid id_j \in ID\}$.

- $\mathsf{Encrypt}(pk_A, M)$: On input the public key pk_A and a message $M \in \mathcal{M}$, the encryption algorithm Encrypt first computes $(K, encKey) = \mathsf{Encapsulate}(pk_A)$. $encData$ is the result of applying the authenticated encryption algorithm to M with key K. Finally, it outputs the ciphertext $C = (encKey, encData)$.

- $\mathsf{Decrypt}(sk_A, C)$: On input the secret key sk_A and a ciphertext $C = (encKey, encData)$, the decryption algorithm Decrypt computes the key $K = \mathsf{Decapsulate}(sk_A, epk)$, and decrypts ciphertext $encData$ using the decryption function of the authenticated encryption scheme to obtain message M if decryption is correct, and $\perp$ otherwise. Finally, it outputs message M (or $\perp$ if decryption was invalid).

- $\mathsf{ReEncFrag}(kFrag, encKey)$: On input a re-encryption key fragment $kFrag$, and an encapsulated key $encKey$, the fragmented re-encryption algorithm ReEncFrag first parses $kFrag = (id, rk)$, and computes $encKey' = (encKey)^{rk}$. Finally it outputs the encapsulated key fragment $F = (encKey', id)$.

- $\mathsf{DecryptFrags}(sk_A, \{F_i\}_{i=1}^{t}, encData)$: On input the secret key sk_A, a set of t fragments of an encapsulated key, each of them labeled as F_i, and the encrypted data $encData$, the fragments decryption algorithm DecryptFrags first computes $encKey' = \mathsf{Combine}(\{F_i\}_{i=1}^{t})$. With this result, it returns the output of $\mathsf{Decrypt}(sk_A, C')$, where $C' = (encKey', encData)$.

Auxiliary Functions

- $\mathsf{Encapsulate}(pk_A)$: On input the public key pk_A, the encapsulation algorithm Encapsulate first randomly generates an ephemeral keypair $(epk, esk) = (g^r, r)$, performs a Diffie-Hellman key agreement between pk_A and esk to compute a shared secret S_A, and uses this shared secret as input to the KDF to produce the key K. The encapsulated key $encKey$ is the ephemeral public key epk. Finally, it outputs $(K, encKey)$.

- $\mathsf{Decapsulate}(sk_A, encKey)$: On input the secret key pk_A, and and encapsulated key $encKey$, the decapsulation algorithm Decapsulate first checks that $encKey$ is a valid public key. Next, it performs a Diffie-Hellman key agreement between $encKey$ and sk_A to compute a shared secret S_A, and uses this shared secret as input to a KDF to produce the key K. Finally, it outputs K.

- $\mathsf{Combine}(\{F_i\}_{i=1}^{t})$: On input a set of t fragments of an encapsulated key, each of them labeled as F_i, the combination algorithm Combine, first parses each fragment F_i as $(encKey_i, id_i)$. Let $I = \{id_i\}_{i=1}^{t}$. Next, it computes the value $encKey'$ as follows:

$$encKey' = \prod_{i=1}^{t} (encKey_i)^{\lambda_{i,I}}, \text{ where } \lambda_{i,I} = \prod_{j=1, j \neq i}^{t} \frac{id_j}{id_j - id_i}$$

Finally, it outputs epk'.

3.1 Parameters of Umbral instantiation in NuCypher KMS

The only restriction that the Umbral cryptosystem imposes on the choice of EC curve is that it should generate a group of prime order, since we need to compute inverses modulo the order of this group.

References

[1] Michael Egorov, MacLane Wilkison, and David Nuñez. Nucypher KMS: decentralized key management system. *CoRR*, abs/1707.06140, 2017.

[2] American National Standards Institute (ANSI) X9.F1 subcommittee. ANSI X9.63 Public key cryptography for the Financial Services Industry: Elliptic curve key agreement and key transport schemes, July 5, 1998. Working draft version 2.0.

[3] M. Blaze, G. Bleumer, and M. Strauss. Divertible protocols and atomic proxy cryptography. *Advances in Cryptology—EUROCRYPT'98*, pages 127–144, 1998.

[4] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[5] Victor Shoup. ISO 18033-2: An emerging standard for public-key encryption. `http://shoup.net/iso/std6.pdf`, December 2004. Final Committee Draft.

[6] IEEE P1363a Committee. IEEE P1363a / D9 — standard specifications for public key cryptography: Additional techniques. `http://grouper.ieee.org/groups/1363/index.html/`, June 2001. Draft Version 9.

[7] V Gayoso Martínez, L Hernández Encinas, and A Queiruga Dios. Security and practical considerations when implementing the elliptic curve integrated encryption scheme. *Cryptologia*, 39(3):244–269, 2015.