FLIC Group Key Proposal

# Summary

There are three actors: the key manager, the publisher, and the consumer. The key manager could be the publisher.

We assume all key pairs are Elliptic Curve from a good curve, such as SECP384R1. They do not all need to be from the same curve. RSA could be used with appropriate algorithm changes, which we describe at the end.

A consumer has a long-term trusted key pair (CSK, CPK) (consumer secret/public key).

A named group has an evolving group key. As users are removed from the group, the group key must evolve. Adding users can be done without changing the group key. We denote this key as (GSK, GPK) (group secret/public key). This is a long-term keypair. The key manger must share the GSK with group members. The published object is signed by the key manager using its trusted long-term key pair (KmSK/KmPK). The key manager may support groups of groups, in which case it must support a method for a user to decide if they are a member of a group by the group’s key id (i.e. sha256(GPK)) and to fetch that key.

A publisher uses a wrapping key pair to share a symmetric key. The wrapping key is usually a short-term (but not emphemeral) EC keypair. We denote this keypair as (WSK, WPK). The wrapping key pair could be used for a period of time, or over several related objects, or for a single manifest. The publisher will encrypt WSK under GPK and publish it where manifest consumers may fetch it. There is a *WrappedKeyName* that is placed in a manifest to resolve this key. The published object is signed by the publisher using its long-term (EC) trusted key pair (PuSK/PuPK).

A publisher creates a master secret symmetric key and derives content encryption keys (CEKs) from the master secret. A CEK is used to encrypt the manifest. If the wrapping key is ECC, we use a 1-ephemeral, 1-secret Diffie-Hellman key exchange (ECDHE). The ECDHE offer is published as part of the root manifest and anyone with WSK can derive the same master secret and subsequent CEKs. Because the published offer is signed by the publisher’s long-term ECC key, this is equivalent to ECDHE-ECDSA.

The key manager could simply publish GSK wrapped (e.g. ECC-) under each user’s public key. Or it could use a proxy re-encryption technique (e.g. [Ateniese] or [Wang]). The method of sharing the GSK is beyond the scope of this document. We assume that all parties have a method to resolve a group name to a group key, and there exists a *GroupKeyName* that can be included with the publisher’s encrypted WSK to resolve the appropriate GSK.

|  |  |
| --- | --- |
| **RSA Keypair** | **Protocol Change** |
| KmSK, KmPK | Use RSA -SHA256-PSS for signatures |
| PuSK, PuPK | Use RSA-SHA256-PSS for signatures |
| CSK, CPK | Use RSA-OAEP with AES-KWP to wrap GSK |
| GSK, GPK | Use RSA-OAEP with AES-KWP to wrap WSK |
| WSK, WPK | Use RSA-OAEP to wrap a random master secret |

A Key Manager (KM) manages groups. It issues each group a long-term (until membership changes).

The KM publishes /key\_manager/somewhere/<group\_name>/<version> with a link to the current key’s namespace: /key\_manager/somewhere/<ID(GPK)>. In CCNx, it also publishes a short-lived /key\_manager/somewhere/<group\_name> that points to the current version. However, the normal lookup scheme is from a manifest up to the group key. The RSA group key is GPK/GSK (public/secret keys).

Each user has a UPK/USK (user public/secret key), and the KM knows the UPK. It publishes these under /key\_manager/somewhere/<ID(GPK)>/<ID(UPK)> as a content object that holds the GSK encrypted under UPK using RSA-OAEP (see RsaKeyWrap() below). We take ID(.) to be SHA256(.). The KM can be the same as the publisher.

If using Proxy Re-encryption (e.g. [Ateniese, “third attempt”][Hohenberger]), the KM creates a re-encryption key pair. It publishes the long-term public key and each user’s long-term re-encryption key (which could be generated on-demand), encrypted for that user. It creates a GSK for the group and performs a second level encryption using the long-term public key. Each user may fetch that single object and re-encrypt it to a level 1 encryption that it can then decrypt with its long-term secret key. As per Ateniese, in this scheme the publisher could create the GSK and make a level 2 encryption using the KM’s public key and publish it under its own namespace and users could decrypt it using their re-encryption key. This avoids the KM knowing the GSK. [Wang] is another possibility.

A Publisher creates a wrapping key pair (WPK/WSK), such as from ECC SECP384R1. It may use it for a specific FLIC manifest or a set of manifests or for a time period. It will publish it under /publisher/elsewhere/<ID(WPK)> as an object that has the WSP wrapped under GPK using RSA-OAEP (see RsaKeyWrap() below). The Content Object also identifies the {ID(GPK), GroupKeyLocator}, where GroupKeyLocator is, for example, /key\_manager/somewhere.

To generate a data key (DK) to encrypt a manifest, the publisher uses a 1-sided ephemeral Diffie-Hellman key exchange. The ECDHE uses WPK public key and the secret key from an ephemeral key pair EPK/ESK to create a shared master secret then uses HKDF-SHA256 to derive a 256-bit AES key for DK. Part of the KDF input is the KeyNum used to identify the particular DK. This allows the publisher to derive multiple DKs from the same master secret wrapped in the root manifest.

If WPK/WSK are an RSA key pair, then generate a random master secret and use RSA-ENC(WPK, master\_secret, RSA-OAEP) instead of the ECDHE method.

In the root manifest, the publisher includes {ID(WPK), WrapKeyPrefix, WrappedKey}. Using the above examples, WrapKeyPrefix is /publisher/elsewhere. WrappedKey = {KeyNum, kex\_msg}, where kex\_msg is part of the output of EccKeyWrap(), and is the pair {curve, EPK} signed under WSK. The rest of the manifest then looks like a PresharedKey encryption. Note that WrappedKey is not actually encrypted, it is an ECDHE offer to anyone with WSK.

In summary:

/key\_manager/somewhere/<group\_name>/<version> => Link to /key\_manager/somewhere/<ID(GPK)> namespace

/key\_manager/somewhere/<ID(GPK)>/<ID(UPK)> => Content Object with RsaKeyWrap(UPK, GSK) signed by KM

/publisher/elsewhere/<ID(WPK)> => Content Object with RsaKeyWrap(GPK, WSK) signed by Publisher

(master\_secret, wrapped\_key) = EccKeyWrap(WSK, WPK)

key\_num = sequence number associated with WSK, or large enough random number

dk = KDF(master\_secret, ‘keynum’ || net\_byte\_order(key\_num))

PresharedKeyCtx = {key\_num, IV, mode}, IV = AES VI [see NIST], mode = aes-gcm-128, aes-gcm-256

Root Manifest : {ID(WPK), WrapKeyPrefix, wrapped\_key, PresharedKeyCtx} signed by Publisher

Other Manifest: {PresharedKeyCtx}

[Ateniese] G. Ateniese, K. Fu, M. Green, and S. Hohenberger, “Improved proxy re-encryption schemes with applications to secure distributed storage,” ACM Transactions on Information and System Security (TISSEC), vol. 9, no. 1, pp. 1–30, 2006.

[Hohenberger] Hohenberger, Susan R., Kevin Fu, Giuseppe Ateniese, and Matthew Green. "Unidirectional proxy re-encryption." U.S. Patent 8,094,810, issued January 10, 2012.

[Wang] Wang, Qiang, Wenchao Li, and Zhiguang Qin. "Proxy Re-Encryption in Access Control Framework of Information-Centric Networks." IEEE Access 7 (2019): 48417-48429.