A real hybrid public key encryption scheme (HPKE – RFC9180)

CS7NS5/CSU44032

stephen.farrell@cs.tcd.ie

https://down.dsg.cs.tcd.ie/cs7053

https://github.com/sftcd/cs7053

Setting

- Encryptor/initiator/client somehow has a copy of a public key for a decryptor/recipient/server and wants to use that to encrypt a possible large message to that decryptor/recipient/server
  - Encrypt “to” a public key
- Key agreement uses ephemeral-static ECDH
- Use well studied key derivation functions
- Bulk encryption should use an AEAD cipher
- We’d like options to authenticate, but also using ECDH
- We’d like to be able to bind in pre-shared symmetric keys (as a hedge against quantum attacks)
- The term “hybrid” here is being used differently from it’s use when we talk about PQ stuff
- RFC9180 defines all the above in ways that can be used in practice
  - Used in TLS/ECH and MLS protocols and others

Modes

- Base – encrypt to public key
- Auth – base + ECDH-based authentication
- PSK – base + binding in pre-shared-keys
- PSK-AUTH – do it all

- Only base mode is really in use so far

Code Points

- We specify which ECDH variant we'll be using using a KEM_ID, e.g. p256, x25519, ...
- We specify how we'll derive symmetric keys using a KDF_ID, e.g. hkdf-sha256, ...
- We specify which AEAD ciper using an AEAD_ID e.g. aes-128-gcm, ...
  - There's a "special" AEAD code point for the exporter case where HPKE is just used to generate a good symmetric key that'll be used by the application outside of HPKE
- Both sides need to agree on the above
- All repesented as 16-bit numbers
  - Each such code point also defines the various field lengths involved e.g. size of encapsulated DH public values, length of secrets
- And we also need the public key and probably some other protocol stuff, e.g. for Encrypted ClientHello (ECH) we define an ECHConfig as...

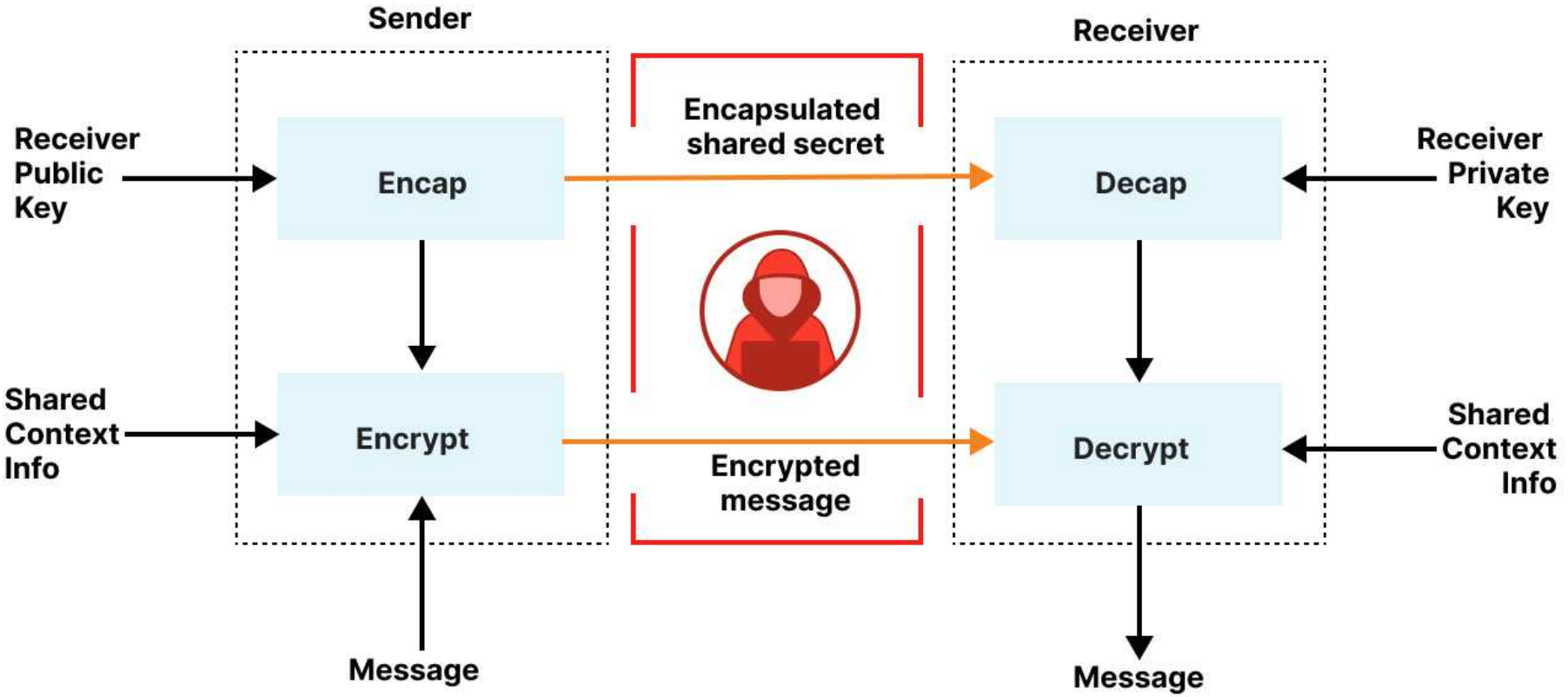
Encrypted ClientHello (ECH) config

```
opaque HpkePublicKey<1..2^16-1>;
uint16 HpkeKemId;  // Defined in RFC9180
uint16 HpkeKdfId;  // Defined in RFC9180
uint16 HpkeAeadId; // Defined in RFC9180
struct {
        HpkeKdfId kdf_id;
        HpkeAeadId aead_id;
} HpkeSymmetricCipherSuite;
struct {
        uint8 config_id;
        HpkeKemId kem_id;
        HpkePublicKey public_key;
        HpkeSymmetricCipherSuite cipher_suites<4..2^16-4>;
} HpkeKeyConfig;
```

```
struct {
        HpkeKeyConfig key_config;
        uint8 maximum_name_length;
        opaque public_name<1..255>;
        Extension extensions<0..2^16-1>;
} ECHConfigContents;
struct {
        uint16 version;
        uint16 length;
        select (ECHConfig.version) {
          case 0xfe0d: ECHConfigContents contents;
        }
} ECHConfig;
```

From https://datatracker.ietf.org/doc/draft-ietf-tls-esni/
A list of ECHConfig structures ends up published in the DNS in an HTTPS resource record for the receiver

HPKE Overview



https://blog.cloudflare.com/hybrid-public-key-encryption/

HPKE comes with some analysis

- One of the reasons to do this was that previous protocols had done similar things in an ad-hoc manner
- HPKE comes with some formal analysis
  - https://www.benjaminlipp.de/p/hpke-cryptographic-standard/
  - https://eprint.iacr.org/2020/1499.pdf
- The hope is that protocols using HPKE will benefit from that analysis so we'll make fewer mistakes in future

OpenSSL API example (1)

```
/* Generate receiver's key pair. */
OSSL_HPKE_keygen(hpke_suite, pub, &publen, &priv,
                                        NULL, 0, NULL, NULL);

/* sender's actions - encrypt data using the receivers public key */
sctx = OSSL_HPKE_CTX_new(hpke_mode, hpke_suite,
                                        OSSL_HPKE_ROLE_SENDER, NULL, NULL);
OSSL_HPKE_encap(sctx, enc, &enclen, pub, publen, info, infolen);
OSSL_HPKE_seal(sctx, ct, &ctlen, aad, aadlen, pt, ptlen);
```

https://github.com/openssl/openssl/blob/master/doc/man3/OSSL_HPKE_CTX_new.pod

OpenSSL API example (2)

```
/* receiver's actions - decrypt data using the private key */
rctx = OSSL_HPKE_CTX_new(hpke_mode, hpke_suite,
                         OSSL_HPKE_ROLE_RECEIVER,
                         NULL, NULL);
OSSL_HPKE_decap(rctx, enc, enclen, priv, info, infolen);
OSSL_HPKE_open(rctx, clear, &clearlen,
               aad, aadlen, ct, ctlen);
```

A few HPKE notes

- People are starting to think now about how to handle PQ algs in HPKE
- Internal data structures process various cryptographic values in “wire” format which affects interop
  - That’s fine, but means those wire formats need to be part of the definition (esp. short/long encoding of nist curve public values)
- Single-shot operation is defined, but not clear if many protocols can benefit from that
  - E.g. HelloRetryRequest in TLSv1.3 means we can’t use single-shot APIs for ECH
- Nonce/sequence handling for multiple calls to seal()/open() is well-defined but re-sync after packet loss on receiver requires using an odd API (`OSSL_HPKE_CTX_set_seq()` in OpenSSL) – I nearly messed up and allowed that for sender’s too!
- OpenSSL test code HPKE is 1694 LOC!

HPKE PR for OpenSSL

- I had some funding to write code for HPKE
- Submitted a PR to OpenSSL in Nov 2021
  - https://github.com/openssl/openssl/pull/17172
- RFC9180 issued in Feb 2022
- Finally merged in Nov 2022
  - Comment from upsteam maintainer: “Wow! What an effort. After nearly one year of work, close to 4.5k net lines-of-code added and >1,300 comments on the PR (not including the other related contribution from @slontis in #19068) - we finally got there”
- I learned quite a lot from that, esp., that it’s hard if your 1st contribution is complex
- I hope to not be the next source of a hearbleed type event (gulp:-)
- My next bit of fun will be trying to do similarly for ECH! (moar gulps:-)

Conclusion

- HPKE seems like a good example of better glue between cryptographic algorithms and protocols/applications
- Thesis: Schemes like this with proofs are better than those invented by one very smart person
- Contributing code for things like this to upstream projects is doable but takes a lot of time, effort and determination
  - And don’t get irritated along the way (or at least, don’t show it:-)