KEM-DEM framework

Tanja Lange

Eindhoven University of Technology

2WF80: Introduction to Cryptology

How to deal with restrictions on the message space?

We can build a hybrid system by public-key encrypting a key for a symmetric system. This might run into issues with the message space.

How to deal with restrictions on the message space?

We can build a hybrid system by public-key encrypting a key for a symmetric system. This might run into issues with the message space.

Shoup suggests to use a key-encapsulation mechanism (KEM) instead.

How to deal with restrictions on the message space?

We can build a hybrid system by public-key encrypting a key for a symmetric system. This might run into issues with the message space.

Shoup suggests to use a key-encapsulation mechanism (KEM) instead.

A key-encapsulation mechanism requires 3 algorithms:

1. Key generation, generating a public-key private-key pair.
2. Encapsulation, taking a public key, producing ciphertext and key.
3. Decapsulation, taking a private key and a ciphertext, producing key.

The key is then used in a data-encapsulation mechanism (DEM).
(This is the regular symmetric-key authenticated encryption.)

How to deal with restrictions on the message space?

We can build a hybrid system by public-key encrypting a key for a symmetric system. This might run into issues with the message space.

Shoup suggests to use a key-encapsulation mechanism (KEM) instead.

A key-encapsulation mechanism requires 3 algorithms:

1. Key generation, generating a public-key private-key pair.
2. Encapsulation, taking a public key, producing ciphertext and key.
3. Decapsulation, taking a private key and a ciphertext, producing key.

The key is then used in a data-encapsulation mechanism (DEM).
(This is the regular symmetric-key authenticated encryption.)

Public-key encryption requires 3 algorithms:

1. Key generation, generating a public-key private-key pair.
2. Encryption, taking a public key and a message, producing ciphertext.
3. Decryption, taking a private key and a ciphertext, producing plaintext.

Turn a PKE into a KEM

Want:

2. Encapsulation, taking a public key, producing ciphertext and key.
3. Decapsulation, taking a private key and a ciphertext, producing key.

Given a public-key encryption system, use the encryption step

2. Encryption, taking a public key and a message, producing ciphertext.

with a random message, i.e., a message sampled uniformly at random from the message space.

Turn a PKE into a KEM

Want:

2. Encapsulation, taking a public key, producing ciphertext and key.
3. Decapsulation, taking a private key and a ciphertext, producing key.

Given a public-key encryption system, use the encryption step

2. Encryption, taking a public key and a message, producing ciphertext.

with a random message, i.e., a message sampled uniformly at random from the message space.

Output the ciphertext as normal and the hash of the message as key.

Turn a PKE into a KEM

Want:

2. Encapsulation, taking a public key, producing ciphertext and key.
3. Decapsulation, taking a private key and a ciphertext, producing key.

Given a public-key encryption system, use the encryption step

2. Encryption, taking a public key and a message, producing ciphertext.

with a random message, i.e., a message sampled uniformly at random from the message space.

Output the ciphertext as normal and the hash of the message as key.

Use the decryption step

3. Decryption, taking a private key and a ciphertext, producing plaintext.

on the ciphertext to get the same message, then hash that message to get the key.

RSA as KEM (skipping key confirmation)

KeyGen (no changes):

1. Pick primes $p, q; p \neq q$.
2. Compute $n = p \cdot q$, $\varphi(n) = (p-1)(q-1)$.
3. Pick $1 < e < n$ with $\gcd(e, \varphi(n)) = 1$.
4. Compute $d \equiv e^{-1} \bmod \varphi(n)$.
5. Output public key (n, e), private key (n, d).

RSA as KEM (skipping key confirmation)

KeyGen (no changes):

1. Pick primes $p, q; p \neq q$.
2. Compute $n = p \cdot q$, $\varphi(n) = (p-1)(q-1)$.
3. Pick $1 < e < n$ with $\gcd(e, \varphi(n)) = 1$.
4. Compute $d \equiv e^{-1} \bmod \varphi(n)$.
5. Output public key (n, e), private key (n, d).

Encapsulation:

1. Pick a random $0 < m < n$
2. Compute $c \equiv m^e \bmod n$.
3. Compute $K = H(m)$.
4. Output (c, K).

Decapsulation:

1. Compute $m' \equiv c^d \bmod n$.
2. Compute $K' = H(m')$.
3. Output K'.

RSA as KEM (skipping key confirmation)

KeyGen (no changes):

1. Pick primes $p, q; p \neq q$.
2. Compute $n = p \cdot q$, $\varphi(n) = (p-1)(q-1)$.
3. Pick $1 < e < n$ with $\gcd(e, \varphi(n)) = 1$.
4. Compute $d \equiv e^{-1} \bmod \varphi(n)$.
5. Output public key (n, e), private key (n, d).

Encapsulation:

1. Pick a random $0 < m < n$
2. Compute $c \equiv m^e \bmod n$.
3. Compute $K = H(m)$.
4. Output (c, K).

Decapsulation:

1. Compute $m' \equiv c^d \bmod n$.
2. Compute $K' = H(m')$.
3. Output K'.

$K' = K$ because RSA ensures that $m' = m$.

Diffie-Hellman as KEM

Everybody knows G and g as well as how to compute in G.
KeyGen:

1. Pick random $0 < a < |G|$.
2. Compute $h_A = g^a$.
3. Output public key h_A, private key a.

Diffie-Hellman as KEM

Everybody knows G and g as well as how to compute in G.
KeyGen:

1. Pick random $0 < a < |G|$.
2. Compute $h_A = g^a$.
3. Output public key h_A, private key a.

Encapsulation:

1. Pick random $0 < r < |G|$.
2. Compute $h = g^r$.
3. Compute $K = H(h_A^r)$.
4. Output (h, K).

Decapsulation:

1. Compute $K' = H(h^a)$.
2. Output K'.

Diffie-Hellman as KEM

Everybody knows G and g as well as how to compute in G.
KeyGen:

1. Pick random $0 < a < |G|$.
2. Compute $h_A = g^a$.
3. Output public key h_A, private key a.

Encapsulation:

1. Pick random $0 < r < |G|$.
2. Compute $h = g^r$.
3. Compute $K = H(h_A^r)$.
4. Output (h, K).

Decapsulation:

1. Compute $K' = H(h^a)$.
2. Output K'.

This works because $h_A^r = g^{ar} = h^a$.