Botan's Implementation of the McEliece PKC

Version 3

Falko Strenzke

July 16, 2021

Document version history		
Version	**Date**	**Remarks**
1	2014	Initial version
2	2021-06-20	Adapted sample code to new API
3	2021-07-16	Added document version history table

1 Introduction

The cryptographic C++ library Botan [1] features an implementation of the McEliece public key cryptosystem (PKC) as of release 1.11.10. The implementation was created by cryptosource GmbH based on the HyMES implementation[2, 3] with kind permission of Nicolas Sendrier and INRIA to release a C++ adaption of their original C code under the Botan license.

The purpose of this document is to give a brief overview of the features of this implementation and provide all information necessary to use the McEliece scheme in an application.

The McEliece scheme is of great importance as a post-quantum cryptographic scheme, i.e. it is thought to be resistant to quantum computers. A quantum computer is a new kind of computer that differs fundamentally from classical computers. It makes use of the laws of quantum mechanics, implying that quantum information theory is the proper framework to describe its computational properties rather than classical information theory. Specifically, quantum computers can execute so-called quantum algorithms, which cannot be run on classical computers. For the world of cryptography this has drastic consequences: there exist quantum algorithms that can break all number theoretic schemes such as RSA and elliptic curve based algorithms.

Today, no practically useful quantum computer can be built due to technological restrictions. However, there are intensive research programs ongoing in various nations. As a matter of fact, nobody is able to predict whether or when the technology will be advanced enough to actually break currently used cryptographic schemes.

However, concerning encryption, like in the application of SSL/TLS, the problem is already pressing today. This is due to the fact that the ciphertexts exchanged today can be recorded and decrypted once a quantum computer is available to the adversary. For signature schemes, the situation is much more relaxed, because it is sufficient to renew existing signatures with a quantum computer secure scheme once the threat of quantum technology becomes real.

For the post-quantum encryption problem, the code-based McEliece scheme provides a number of advantages over other post-quantum schemes. It is very fast in encryption and decryption, key generation times are also acceptable. It is patent-free, and the the syndrome decoding problem, which this scheme is based on, is quite well investigated – because it is very meaningful in the area of error correcting codes with vast applications in the industry since the start of the electronic communications era.

The great drawback of this scheme is its huge public key size. This probably rules this scheme out as the general successor of RSA, since it poses a great problem for resource-constrained devices. However, to achieve security in the field of conventional computers, the McEliece scheme is a useful ad hoc solution to the quantum computer problem.

2 McEliece in Botan

The McEliece implementation in Botan, is, as already stated, based on the HyMES [2, 3] C implementation. The major modifications to the scheme are

- removal of the constant-weight-word encoding, which was used in HyMES to to encode additional information in the error vector,

- exchange of the algorithm used to determine the roots of the error locator polynomial: while HyMES uses the Berlekamp trace algorithm, Botan uses one based on polynomial decompositions [4]. This is due to efficiency as well as side-channel security considerations.
- implementation of the side-channel countermeasures from [4, 5],
- removal of some fault-attack vulnerabilities [4],
- a countermeasure against the side-channel attacks presented in [6]
- implementation of key encapsulation method (KEM) which provides security against adaptive chosen ciphertext attacks when the payload is encrypted with authenticated encryption.

The McEliece scheme may never be used as a raw scheme. This is for two reasons: first, this scheme is heavily vulnerable to adaptive chosen ciphertext attacks. This makes it necessary to wrap the scheme in a so-called CCA2-conversion (CCA2-security is the technical term for adaptive chosen ciphertext security). As a consequence, this makes possible a modification to the originally published scheme: the public matrix can be brought into systematic form, i.e. a part of that matrix is the identity matrix. Both the original HyMES and the Botan implementation use this technique. This leads to the second reason why not to use this scheme without a CCA2 conversion: the identity matrix causes the message to be reproduced only slightly distorted within the ciphertext. Thus, using scheme in this configuration without a conversion would even make the message visible in the ciphertext to a large extend.

3 KEM scheme

In the Botan implementation, the CCA2-security is reached by the use of a KEM scheme. In such a scheme, rather than encrypting a specified message directly with the public key scheme, the encryption algorithm creates a random message from which it derives a symmetric key which is then used to encrypt the payload data. The decryption algorithm correspondingly returns the same symmetric key – provided that ciphertext was not manipulated.

The encryption and decryption of our scheme is realized as shown in Algorithms 1 and 2. Note that in these algorithms the concatenations are performed on bit strings that are padded up to lengths that are multiples of eight before the concatenation. "KDF" denotes a key derivation function that has to specified for a concrete instantiation of the scheme. One possibility of a strong KDF would be "KDF1 with SHA512"[1], which is available in Botan.

The security of a corresponding KEM scheme has been proven for the closely related Niederreiter PKC [7], and the proof should be naturally extendible to the McEliece scheme.

Algorithm 1 McEliece - KEM encryption

Require: public key $\mathbf{G}^{\text{pub}}$, code parameters n (code length) and t (error weight)
Ensure: ciphertext $\vec{z} \in \mathbb{F}_2^n$ and symmetric key $K \in \mathbb{F}_2^{512}$
$\vec{u} \leftarrow$ random k-bit string // $k = n - mt$, $m = \lceil \log_2(n) \rceil$
$\vec{e} \leftarrow$ random error vector of length n and weigth t
$(\vec{z}, \vec{e}) \leftarrow \mathcal{E}_{\mathbf{G}^{\text{pub}}}(\vec{u})$ // McEliece encryption
$K = \text{KDF}(\vec{u} || \vec{e})$
return $\vec{z}$, K

[1] Standardised in ISO-18033-2

Algorithm 2 McEliece - CCA2 secure decryption

Require: ciphertext $\vec{z} \in \mathbb{F}_2^n$ secret key S
Ensure: symmetric key $K \in \mathbb{F}_2^{512}$
$(\vec{u}, \vec{e}) \leftarrow \mathcal{D}_S(\vec{z})$ // McEliece decryption
$K = \mathrm{KDF}(\vec{u}||\vec{e})$

n	t	**security level in bits**
1632	33	80
2480	45	107
2960	57	128
3408	67	147
4624	95	191
6624	115	256

Table 1: A selection of parameters for the McEliece scheme with the corresponding classical security level.

Security against adaptive chosen ciphertext attacks is achieved by using the key K output by the encryption and decryption algorithms to encrypt/decrypt the data using authenticated encryption (AE). Botan provides a number of AE schemes which can be used for this purpose: EAX, OCB, GCM, SIV, and CCM.

4 Parameter Choice

As a code-based scheme, unlike number theoretic schemes like RSA and elliptic curve based schemes, a McEliece private key is not determined by single parameter, but two: the code length n and the error weight t. We give a number of parameters sets with corresponding approximate key security levels in Table 1. These parameters, constructed to minimize the respective public key size for a given security level, are taken from [8]. For all of these parameter sets, the security level is up to few bits lower than indicated in our table, since the authors of [8] assume the addition of more than t errors which is not supported by the Botan implementation.

To achieve security of against a quantum computer adversary of a given level (quantum binary work factor), a classical parameter set with the double classical security level has to be chosen [9]. I.e., to have 128-bit security level against quantum attacks, one has to choose parameters which provide 256-bit security against classical attacks.

5 Using the Botan's McEliece Implementation

The following example code shows the McEliece key generation, encryption and decryption using the above described KEM scheme. Note that it is an artefact of Botan's API for KEM schemes that also the decryptor receives an RNG. It is not used in the McEliece decryption.

```
1
2  #include <botan/mceliece.h>
3  #include <botan/pubkey.h>
4  #include <botan/system_rng.h>
5  #include <iostream>
6  int main()
```

```
 7  {
 8
 9    Botan::u32bit n = 6624;
10    Botan::u32bit t = 115;
11    // at the receiver's side:
12    Botan::System_RNG rng;
13    Botan::McEliece_PrivateKey mce_priv(rng, n, t);
14    Botan::secure_vector<Botan::byte> encoded_private_key = mce_priv.
          private_key_bits();
15    std::unique_ptr<Botan::Public_Key> mce_pub = mce_priv.public_key();
16    std::vector<Botan::byte> encoded_public_key = mce_priv.public_key_bits();
17    // send the public key to the sender
18    //
19    // at the sender's site:
20    Botan::McEliece_PublicKey decoded_mce_pub(encoded_public_key);
21    Botan::PK_KEM_Encryptor kem_enc(decoded_mce_pub, rng, "KDF1(SHA-512)");
22    Botan::secure_vector<uint8_t> encap_key; // this has to go over the wire
23    Botan::secure_vector<uint8_t> prod_shared_key; // this is the key used for
          encryption of the payload data
24    kem_enc.encrypt(encap_key, prod_shared_key, 64, rng);
25
26    // [...] encrypt payload data with shared key => encrypted payload
27    // [...] send encap_key and encrypted payload to receiver
28
29    // at the receiver's site:
30    Botan::McEliece_PrivateKey decoded_mce_priv(encoded_private_key);
31    Botan::PK_KEM_Decryptor kem_dec(decoded_mce_priv, rng, "KDF1(SHA-512)");
32    Botan::secure_vector<uint8_t> dec_shared_key = kem_dec.decrypt(encap_key.
          data(), encap_key.size(), 64);
33    // [...] decrypt the payload data with the dec_shared_key
34
35    // test:
36    if(dec_shared_key != prod_shared_key)
37    {
38      std::cerr << "error␣decrypting␣shared␣key" << std::endl;
39    }
40    return 0;
41  }
```

References

[1] The Botan Library: botan.randombit.net.

[2] Biswas, B., Sendrier, N.: HyMES - an open source implementation of the McEliece cryptosystem (2008) http://www-rocq.inria.fr/secret/CBCrypto/index.php?pg=hymes.

[3] Biswas, B., Sendrier, N.: McEliece Cryptosystem Implementation: Theory and Practice. In: PQCrypto. (2008) 47–62

[4] Strenzke, F.: Fast and secure root finding for code-based cryptosystems. In Pieprzyk, J., Sadeghi, A.R., Manulis, M., eds.: Cryptology and Network Security, CANS 2012. Volume 7712 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2012) 232–246

[5] Shoufan, A., Strenzke, F., Molter, H., Stöttinger, M.: A Timing Attack against Patterson Algorithm in the McEliece PKC. In Lee, D., Hong, S., eds.: Information, Security and Cryptology - ICISC 2009. Volume 5984 of Lecture Notes in Computer Science., Springer Berlin / Heidelberg (2009) 161–175

[6] Strenzke, F.: Timing Attacks against the Syndrome Inversion in Code-Based Cryptosystems. In Gaborit, P., ed.: Post-Quantum Cryptography. Volume 7932 of Lecture Notes in Computer Science., Springer Berlin Heidelberg (2013) 217–230

[7] Persichetti, E.: Secure and Anonymous Hybrid Encryption from Coding Theory. In Gaborit, P., ed.: Post-Quantum Cryptography. Volume 7932 of Lecture Notes in Computer Science., Springer Berlin Heidelberg (2013)

[8] Bernstein, D.J., Lange, T., Peters, C.: Attacking and defending the McEliece cryptosystem. Post-Quantum Cryptography, LNCS **5299** (2008) 31–46

[9] Bernstein, D.: Grover vs. McEliece. In: The third international Workshop on Post-Quantum Cryptography PQCRYPTO 2010, Lecture Notes in Computer Science