How to implement the public Key Operations in Code-based Cryptography on Memory-constrained Devices

Falko Strenzke[1]

[1] FlexSecure GmbH, Germany**,
strenzke@flexsecure.de
[2] Cryptography and Computeralgebra, Department of Computer Science,
Technische Universität Darmstadt, Germany

Abstract. While it is generally believed that due to their large public key sizes code based public key schemes cannot be conveniently used when memory-constrained devices are involved, we propose an approach for Public Key Infrastructure (PKI) scenarios which totally eliminates the need to store public keys of communication partners. Instead, all the necessary computation steps are performed during the transmission of the key. We show the feasibility of the approach for a concrete example platform and analyze a number of code-based cryptographic schemes with respect to its applicability.

Key words: post-quantum cryptography, code-based cryptography, public key encryption scheme, efficient implementation, embedded devices

1 Introduction

Code-based cryptography, i.e. the class of cryptographic schemes build on error correcting codes, encompasses public key encryption schemes [1, 2] as well as signature schemes [3, 4] and an identification scheme [5]. The main advantage of code-based cryptographic schemes over currently used schemes that are based on factoring or elliptic curves is their security in the presence of quantum computers [6], but at least the encryption schemes' operations can also be implemented comparatively fast [7]. However, the large public key size in these schemes are considered a tremendous disadvantage. For this reason, a number of attempts have been made to reduce the key size by using special codes [8, 9]. But some of these attempts have already been shown to result in insecure cryptosystems [10]. More recent proposals, for instance [11], will have to prevail for some time until they can be granted the same trust as the original McEliece, proposed 32 years ago, which uses classical binary Goppa codes [12] and is still regarded as secure.

In this work, we address the problem of performing the public operations, i.e. encryption or signature verification, of code-based public key cryptosystems

** A part of the work of F. Strenzke was done at[2]

on devices with limited memory resources, like for instance smart cards. Typically, smart cards have less than 20 KB of RAM, while the available amount of non-volatile memory (NVM), e.g. flash-memory, can be as large as 512 KB [13, 14]. If a public key of a communication partner shall be temporarily stored on the device for the purpose of performing e.g. an encryption, it would have to be stored in the NVM since it exceeds the size of the RAM many times over. Specifically, the public keys will be at least 100 KB large for reasonable security parameters, as we will see in Section 2.3. For instance, the works [15–17] all describe implementations of code-based encryption schemes on embedded devices, where the public key is stored in the devices NVM. The drawbacks of storing such an amount of data in the device's NVM are first of all the cost of keeping such a large amount of memory available for this purpose and also the much slower writing speed compared to RAM access. In order to circumvent these problems, we show in this work that the public operations can be executed by only storing very small parts of the public keys at any given time during the operation. Our approach also considers that these operations are always carried out in a PKI context, which implies the verification of user public key certificates against issuer certificates.

The paper is organized as follows. In Section 2 we give the preliminaries about PKI and code-based encryption schemes needed for the remainder of the paper. The newly proposed approach is introduced in Section 3 and it's concrete computational complexity is analyzed. Subsequently, in Section 4, we show the feasibility of the proposed approach on a concrete platform based on the analysis of the preceding section. Two possible variants to the presented approach are given in Section 5. Next, in Section 6 we show the possibility of treating code-based signature schemes in the same way as the encryption schemes which we focus on in this work. Section 7 addresses the applicability of the approach to the code-based identification scheme proposed by Stern [5].

2 Preliminaries

2.1 Public Key Cryptography

In a public key infrastructure, the trustworthiness of a public key is always verified against a trust anchor. From the trust anchor, which is usually a certification authority (CA) certificate, to the user certificate, there is a certificate chain involved. The trustworthiness of a certificate lower in the chain is guaranteed by its authentic digital signature created by the respective issuer, verifiable via the corresponding public key contained in the issuer certificate.

For the case of public key encryption, it means that a user A's public key intended for encryption is contained in the user certificate. A user B willing to encrypt a message for A thus goes through the following steps:

1. retrieve A's public encryption certificate Enc-Cert_A (for example by accessing a database or asking A directly)

2. verify the authenticity of Enc-Cert_A by checking the signature on the certificate against the trust anchor (CA certificate)
3. encrypt the secret message using Enc-Cert_A and send it to A

Since in this work we will address problems and solutions for embedded devices such as smart cards, we wish to point out why it is necessary to be able to carry out not only the private operations of a public key scheme (i.e. decryption or signature generation) but also the public operations on such devices. One application are key exchange schemes. Key exchange schemes based on public key cryptography are used for instance in the context of the German ePassport [18]. There, an elliptic curve based key agreement scheme is realized [19]. In order to replace this scheme with a quantum computer secure solution, one would have to combine a public key encryption scheme with a public key signature scheme that both have this property. Then, one party sends the signed and encrypted symmetric key to the other party. In the mentioned context this means that eventually the ePassport's chip has to carry out the encryption operation.

2.2 Linear Error Correcting Codes

In this section we briefly explain the basics of linear error correcting codes as needed for the understanding of the subsequent sections. A linear binary error correcting code $\mathcal{C}$ is a set of code words $\{c_i\}$ and is characterized by

- the code size n, which defines the bit length a code word c_i of the code $\mathcal{C}$,
- the code dimension k with $k < n$, which defines the bit length of the message words v that can be encoded,
- the error correcting capability t, which is the number of bit flips that may be applied to a code word c and still allow recovering the corresponding message v.

The encoding of a message v is performed by multiplying it by a generator matrix $G \in \mathbb{F}_2^{k \times n}$, which is specific for this code: $c = vG$.

The decoding is performed by first multiplying the eventually distorted code word c' by the so called parity check matrix $H \in \mathbb{F}_2^{(n-k) \times n}$, also being specific for the respective code: $s = Hc'^T$, where $s \in \mathbb{F}_2^{n-k}$ is called the syndrome of the distorted code word c'. Given that no more than t bit flip errors occurred, i.e. the hamming distance between c and c' is less or equal than t, a decoding algorithm can be applied to recover the message v. The decoding algorithm is based on the underlying code and receives s as input.

2.3 Code-based Encryption Schemes

In the following, we explain two code-based encryption schemes, where we focus on the encryption operation, since the details of decryption operation are irrelevant to the subject of this work.

The first encryption scheme we present is the McEliece [1] scheme. The McEliece public key consists of the so called public generator matrix G_p. Algorithm 1 shows the McEliece encryption operation. The idea behind the McEliece scheme is that the holder of the corresponding private key knows a secret error correction code $\mathcal{C}_s$ that allows him to recover the error vector e and find the message m. Specifically, the public key is given by $G_p = TG_sP$, where $T \in \mathbb{F}_2^{k\times k}$ is a random invertible matrix, P is a random $n \times n$ permutation matrix and G_s is the generator matrix of a secret code, all of which are parts of the private key. The decoding works since we find that applying the inverse permutation to the ciphertext gives

$$z' = zP^{-1} = \underbrace{mTG_s}_{\in \mathcal{C}_s} + \underbrace{eP^{-1}}_{e'}.$$

Since the first term on the right hand side is a code word of $\mathcal{C}_s$ and the second term is the permuted error vector having hamming weight t, the message m can be recovered using the error correction algorithm. On the other hand, without the knowledge of the secret key, recovering the message m is intractable for secure parameters.

Algorithm 1 The McEliece encryption Operation

Require: the McEliece public key $G \in \mathbb{F}_2^{k\times n}$ and the message $m \in \mathbb{F}_2^k$,
Ensure: the ciphertext $z \in \mathbb{F}_2^n$
create a random binary vector $e \in \mathbb{F}_2^n$ with Hamming weight $\mathrm{wt}\,(e) = t$
$z \leftarrow mG_p \oplus e$

The McEliece parameters are given by the code parameters n, k and t. An example parameter set giving about 100 bits of security with respect to the attacks given in [20] would be be $n = 2048$, $k = 1498$ and $t = 50$.

In order for the scheme to be secure against chosen ciphertext attacks, a so called CCA2-conversion has to be applied to the scheme [21]. Given such a CCA2-conversion is used, it is also possible to choose the matrix T in such a way that G_p is in systematic form, i.e. $G_p = [\mathbb{I}_k|R]$, where $\mathbb{I}_k$ is the $k \times k$ identity matrix. Then, for the parameter set mentioned above, the public key can be represented by $R \in \mathbb{F}_2^{k\times(n-k)}$, which has a size of about 100 KB.

The other encryption scheme is the Niederreiter [2] scheme. Here, the public key consists of the public parity check matrix $H_p = TH_sP$, where H_s is the parity check matrix of the private code and $H_p \in \mathbb{F}_2^{(n-k)\times n}$, and T and P are chosen equivalently to their counterparts in the McEliece scheme. Furthermore, as in the McEliece scheme, H_p can be put in systematic form. Then the public key will be of the same size as for the McEliece cryptosystem. The Niederreiter encryption is depicted in Algorithm 2. The message is encoded into an error vector of weight t and the ciphertext is the corresponding syndrome, which can only be decoded by the holder of the private key.

Algorithm 2 The Niederreiter encryption Operation

Require: the Niederreiter public key $H \in \mathbb{F}_2^{(n-k)\times n}$ and the message m
Ensure: the ciphertext $z \in \mathbb{F}_2^{n-k}$
encode the the message m into $e \in \mathbb{F}_2^n$, where $\mathrm{wt}(e) = t$, using an appropriate algorithm ("constant-weight-word encoding")
$z \leftarrow eH$

3 Online Public Operation

In this section, we explain the main idea of the paper, namely how to implement the public operations of code-based schemes without storing full public keys on the device. In a straightforward approach, the public operation, which we here assume to be an encryption operation, would be realized by first retrieving the public key (embedded into a public key certificate containing also a signature) of the communication partner, storing it on the device, computing the hash value of the certificates to-be-signed (TBS) data (which includes the code-based public key), verifying the signature, and finally encrypting the designated message using the certificate's public key. With the proposed approach however, no storage of the whole public key is required. Instead, only a comparatively small amount of RAM memory will be used. The basic idea is to use the computation time that is available to the devices CPU in the time interval between the receival of two bytes via the serial interface. During this interval both the encryption algorithm and the hash algorithm are advanced by one small step. Hence we call this approach "online public operation".

This approach works because both the computation of the hash value of the public key and the matrix-vector product only depend on a small part of the whole public key at any given point in time: while the hash function acts on blocks of multiple bytes (for instance 64 bytes for SHA-256), the matrix multiplication could in principle be carried out bit-wise.

In Figure 1, the complete process of the online public operation approach is depicted. On the left hand side, the processing of the certificate containing the code-based public key to be used in the public operation is shown. Here, we assume that the public key is contained in an X.509 public key certificate [22]. Such a certificate is constituted by the sequence of the TBS data, followed by a field containing information about the signature algorithm (not shown in the figure) and finally the signature. The signature ensures the authenticity of TBS data, and is calculated based on their hash value, using a hash algorithm as specified in the preceding information field. Please note that the signature algorithm used to sign the user certificate needs not to be code-based (in which case the trust anchor certificate would contain a large code-based key itself). Instead, a hash based signature scheme [23] could be used. These schemes are also quantum computer resistant and feature extremely small public keys.

In Step 1a the part of the TBS data that precedes the public key is received by the device and processed in the normal manner, which includes the computation of the hash value of the received data. Once the transmission of the public key,

i.e. the public matrix M, begins (Step 2a), the computation of the product vM begins, where v is a binary vector whose meaning depends on the type of the code-based scheme. In an encryption scheme like McEliece or Niederreiter, v represents a message. The hash computation is also continued. After the whole public matrix has been received, the remaining TBS data is again processed in the normal manner (Step 3a). Finally, when the TBS data have been completely received the hash value of the TBS data is ready. It is then used to verify the certificate's signature with the help of the certificate of the issuer I which is stored on the device as the trust anchor (Step 4).

The public operation of the code-based scheme is potentially composed of computations before the matrix-vector product is needed (Step 1b). These computations can be done before the public matrix transmission begins, e.g. they could be carried out before and/or during the receival of the TBS data preceding the public key. Once the public key matrix has been fully received and processed (i.e. after Step 2a), the remaining computations of the public operation are carried out (Step 3b), e.g. the addition of the error vector e in the McEliece scheme. The result is either a ciphertext (in case of an encryption scheme) or a Boolean value (in case of a signature verification). But whether this result is output respectively further processed by the device (Step 5a) depends on the result of the signature verification (Step 4). If the verification fails, the device will output an error answer (Step 5b).

In the following subsections, we take a closer look at the actual operations that have to be carried out after the receival of a single byte. This analysis will build the basis for the evaluation of the expected performance on specific platforms.

3.1 The Matrix-Vector Multiplication

The Matrix Vector Multiplication $b = aM$, with $a \in \mathbb{F}_2^{l_a}$, $b \in \mathbb{F}_2^{l_b}$ and $M \in \mathbb{F}_2^{l_a \times l_b}$ can be realized in two different ways: the matrix can be processed column-wise or row-wise. In this section we will only consider the former solution. A justification for this choice as well as a consideration of the second variant will be given in Section 5.1.

In the column-wise multiplication, each bit of b is computed as $b_i = \sum_j m_j M_{ij}$. Assuming byte-wise operations, 8 result bits of the product $m_j M_{ij}$ can be computed simultaneously by performing the bit-wise logical AND of a byte containing a part m and another byte containing the corresponding part of the i-th column of M. The resulting byte is logically XORed with a state byte. The result is the new value of this state byte, which of course has value zero at the start of each matrix column. After the last byte of the i-th column has been received and processed in the way described above, b_i is computed as the parity bit of the state byte. The parity bit of a byte can be computed in 6 instructions [24]. Together with the operations that have to be carried out for each byte, processing the last byte of a matrix column can be done in roughly 8 instructions.

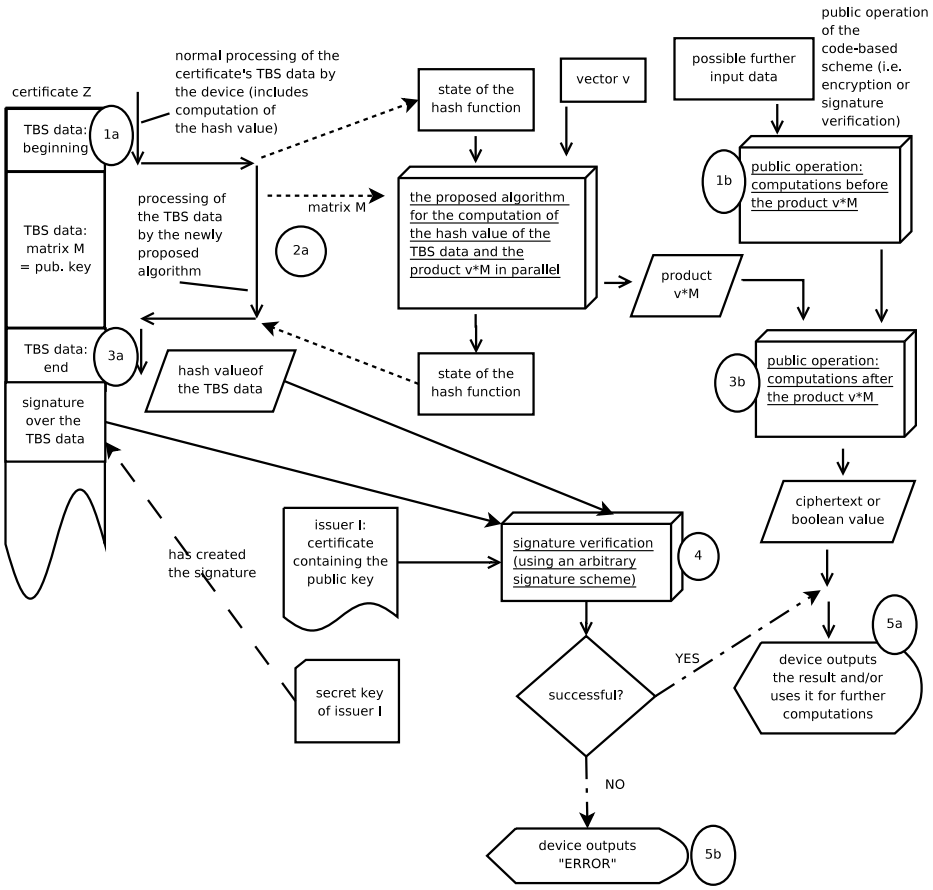

Fig. 1. Overview of the complete process of the online public operation.

3.2 Partitioned Computation of the Hash Value

For the hash value computation, two different approaches are conceivable. Firstly, the compression function, which processes a single block of input data, can be split into a number of equally complex parts, where in the ideal case the number of these parts is equal to the number of bytes in one block. Then, after the receival of each byte one such part of the compression function is carried out. The other approach would be to use a continuous hash computation which operates asynchronously to the receiving unit and the matrix-vector multiplication.

In the following, we will only follow the first approach, since it turns out that for the prominent SHA-256 hash function [25] we find that the above mentioned ideal partitioning is possible.

The SHA-256 compression function processes 64 byte blocks of input data one after another. It features 64 iterations of the step function, which operates on the block of input data and the hash functions internal state. Thus it suffices to carry out one iteration of the step function between the receival of two bytes, where the last completely received block is processed.

A superficial count of the 32-bit instructions needed in one of these 64 iterations of the step function gives 32 (provided that register rotation instructions are available on the platform). This fits quite well with measurement results for SHA-256 on a Pentium 4 given in [26].

The RAM demands of this solution are 160 bytes: a 64 byte receive buffer to store the currently transmitted block, another 64 byte buffer for the currently processed block and 32 bytes for the internal state of the SHA-256 hash function.

4 Consideration of Transmission Rate and Computation Speed

In this section, we apply the considerations of the previous section to a specific platform. Namely, we regard an SLE66CLX360PE [14] smart card platform from Infineon Technologies AG. It features an ISO/IEC 14443 compliant contactless interface which can transmit up to 106 KB/s. This allows the transmission of a McEliece public key of size 100 KB for the parameters given in Section 2.3 in about 1s, which can be considered at least acceptable for certain applications.

Let us now consider whether the computational power of this device is able to support this transmission speed, i.e. whether it is able to perform all the necessary operations for the hash value computation and the matrix-vector multiplication during the receival. The device features a 16-bit CPU which runs at up to 30 MHz. Due to the fact that our above considerations about the number of instructions that have to be executed after the receival of each byte are based on a 32-bit architecture and that we did not take into account any overhead like the increment of loop variables, we multiply our estimate of 40 instructions by 5, which surely takes us to the safe side. We also assume that each instruction takes one clock cycle. According to the calculation $106 \cdot 1024 \cdot 40 \cdot 5$ cycles$/s \approx 21 \cdot 10^6$ cycles$/s$, it is sufficient if the CPU runs at

21 MHz in order to perform the necessary operations. Even if this estimation should turn out to be still too optimistic also considering the remaining 9 MHz reserve, it should be clear that at least a 32-bit platform running at the same frequency will fulfill the requirements.

In the future, contactless transmission rates may be about 8 times higher [27] than the rate considered above. It is still feasible to support such a high transmission rate at a CPU speed of 30 MHz if adequate hardware support is available on the device, since in this case there are still about 3.5 cycles available between the receival of two bytes. Thus, in this future scenario the running time of the matrix multiplication might be even further decreased by a factor of 8.

5 Variants of the proposed Approach

5.1 Column-wise vs. Row-wise Matrix-Vector Multiplication

As stated in Section 3.1, row-wise computation of the matrix-vector multiplication is an alternative to the column-wise approach. In this case the computation of the result is according to $b = \sum_j M_i a_i$, where M_i is the vector represented by the i-th row of M. This means that a row M_i is added to the result if the corresponding bit a_i is one, otherwise nothing has to be done. In the normal case, where the whole matrix is available instantly, this approach has a significant advantage over the column-wise approach since the vector a will contain a large number of zero bits. But in the case of the online public operation, this advantage disappears since the matrix-vector multiplication's running time is determined by the transmission time alone (under the assumption of sufficient computational power of the device as analyzed in Section 4). The row-wise approach would only have an advantage if the saved computational effort could be used to perform other tasks, which can be assumed to be rather unlikely or at least of minor relevance in the context of embedded devices such as smart cards.

On the other hand, the disadvantage of the row-wise multiplication lies in its potential side-channel vulnerability. Specifically, if an attacker is able to find out whether the currently transmitted row is added or ignored, for instance by analyzing the power trace [28], he can deduce the value of the secret bit a_i. Of course, countermeasures can be implemented. A certain randomization could for instance be introduced by keeping a number of received rows in a buffer and processing them in a randomized order. However, whether the questionable computational advantage of this method is worth such efforts must be decided in a concrete implementation scenario.

In any case, once the X.509 key format for a code-based scheme is defined, the choice for one of the two methods is taken. While it then would still be possible to transmit the matrix in the other orientation in order to carry out the multiplication, the online hash computation only works if the correct orientation is used.

5.2 Using non-binary Codes

In in the original papers, only binary Goppa codes are used in the McEliece and Niederreiter scheme. Recently, the employment of codes over an alphabet $\mathbb{F}_q$ have also been considered [29]. The methods proposed in this work are also applicable in this case. It must be pointed out however, that the multiplication of the matrix and vector elements becomes more complex for codes over larger alphabets, eventually calling for more computational power at high transmission rates.

6 Code-based signature Schemes

A number of code-based signature schemes have been proposed. In the following, we will address two of these schemes very briefly with the goal of showing that the proposed approach for the online public operation is applicable to both of them.

In [3], the McEliece scheme is inverted in the sense that the signer proves his ability to decode a binary vector related to the message using a certain code. Thus, the signature verification basically consists of a matrix-vector multiplication just like for the encryption schemes described in Section 2.3.

A signature scheme involving two binary matrices as the public key is presented in [4]. In the verification operation, both matrices have to be multiplied by a vector. Thus the online public operation can be carried out by transmitting them one after another.

7 The Stern identification Scheme

In [5], an identification scheme based on coding theory is proposed. The goal of this section is to find out whether the proposed online public operation can be applied to this scheme. We first describe the scheme, where we do not give any considerations of it's security.

In the Stern identification scheme, there are public parameters that are given by a parity check matrix $H \in \mathbb{F}_2^{(n-k)\times n}$ and an integer $w < n$ that are shared by a group of users. The choice of the security parameters n and k is taken analogously to that of the parameters for code-based encryption schemes. Each user has a private key $s \in \mathbb{F}_2^n$ with weight w and publishes his public key $i = Hs^T$. Furthermore, a cryptographic hash function $h()$ is involved. The identification scheme allows A to prove to B that she knows the secret s without divulging any information about it. It consists of r rounds, each round is defined by:

1. A (the prover) chooses randomly a word $y \in \mathbb{F}_2^n$ and a permutation σ of $\{1, 2, \ldots, n\}$. A sends to B the commitment c_1, c_2, c_3 such that:

   $$c_1 = h(\sigma | Hy^T); c_2 = h(\sigma(y)); c_3 = h(\sigma(y \oplus s))$$

   where $\sigma(x)$ denotes the image of x under the permutation σ and "|" the concatenation.

2. B (the verifier) sends to A a random challenge $b \in \{0, 1, 2\}$
3. depending on b, A reveals certain values to B:
   (a) if $b = 0 : A$ reveals y and σ
   (b) if $b = 1 : A$ reveals $(y \oplus s)$ and σ
   (c) if $b = 2 : A$ reveals $\sigma(y)$ and $\sigma(s)$
4. B verifies a subset of the commitment depending on b:
   (a) if $b = 0 : B$ checks that c_1 and c_2 are correct by computing $c'_1 = h(\sigma|Hy^T)$ and $c'_2 = h(\sigma(y))$
   (b) if $b = 1 : B$ checks that c_1 and c_3 are correct by computing $c'_1 = h(\sigma|Hy^T) = h(\sigma|H(y \oplus s)^T \oplus i)$ and $c'_3 = h(\sigma(y \oplus s))$
   (c) if $b = 2 : B$ checks that c_2 and c_3 are correct by computing $c'_2 = h(\sigma(y))$ and $c'_3 = h(\sigma(y) \oplus \sigma(s))$ and verifies that $\sigma(s)$ is of weight w

The cheating probability for a single round is $2/3$. If a number r of rounds is carried out, then the cheating probability is reduced to $(2/3)^r$. For instance, 30 rounds can be used to reduce the cheating probability below 2^{-17}. The scheme can be implemented quite efficient on resource constrained devices as demonstrated in [30].

It is immediately obvious that the online multiplication can be applied to the operations the prover (A) has to carry out in each round, i.e. the multiplication Hy^T. This removes the need to store the huge matrix H on the device and instead store only its hash value. Since the t rounds are all independent of each other, it is also possible to compute Hy^T for a certain number of rounds $u \leqslant r$ in parallel. Then the matrix H would have to be transmitted for any such group of parallel computed rounds, thus u should be chosen as large as possible for an efficient implementation. But for a large number of rounds the memory demands become considerable: at the moment the u-fold parallel matrix multiplication finishes, the prover has u sets $\{Hy^T, \sigma, y\}$ in his memory. Especially the permutation σ is of considerable size: it is represented in $\log_2(n) \cdot n$ bits, which amounts to 2816 bytes for the example code parameter n from Section 2.3.

For the verifier (B), it is also possible to apply the online multiplication to a number of rounds in parallel, where he has to carry out up to to two multiplications in each round, depending on the respective challenge b. But the protocol requires the verifier to store two commitments for each of the u rounds carried out in parallel, until he receives the corresponding revelations according to the respective value of the challenge b. Also then, if the matrix H is not stored on his device but input after the revelations have been received, all those revelations of the u rounds have to be stored until the receival of H.

Thus for this scheme it seems more appropriate to store the matrix H on the device. This is not a large drawback, since this matrix is not an individual public key but a parameter shared by a whole group of users and can thus be kept on the device permanently. Also note that the private key in this scheme only has a bit length given by the code parameter n, e.g. consumes 256 bytes for the example code parameters given in Section 2.3, which yields about the same NVM demands as for the McEliece encryption scheme where a parity check matrix is part of the private key (at least if fast decryption shall be possible).

8 Conclusion

In this work we have shown an approach for implementing the operations involving code-based public keys on memory-constrained devices like smart cards, that covers the matrix-vector multiplication as well as the hash computation for the verification of the user certificate. The solution is applicable to basically all code-based encryption and signature schemes that have been proposed so far. Thus we are confident that this work improves on the applicability of this class of cryptographic schemes by reducing the impact of the large public key sizes for memory-constrained devices.

Especially the Niederreiter encryption scheme becomes attractive when used with the proposed online public operation. This is because as demonstrated in [17], for this scheme also the private key can kept well below 10 KB for reasonable parameters, enabling both encryption and decryption on devices with small RAM and NVM.

References

1. McEliece, R.J.: A public key cryptosystem based on algebraic coding theory. DSN progress report **42–44** (1978) 114–116
2. Niederreiter, H.: Knapsack-type cryptosystems and algebraic coding theory. In: Problems Control Inform. Theory. Volume Vol. 15, number 2. (1986) 159–166
3. Courtois, N., Finiasz, M., Sendrier, N.: How to Achieve a McEliece-Based Digital Signature Scheme. In Boyd, C., ed.: Advances in Cryptology - ASIACRYPT 2001. Volume 2248 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg (2001) 157–174
4. Kabatianskii, G., Krouk, E., Smeets, B.: A digital signature scheme based on random error-correcting codes. In Darnell, M., ed.: Crytography and Coding. Volume 1355 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg (1997) 161–167
5. Stern, J.: A new identification scheme based on syndrome decoding. In: CRYPTO '93: Proceedings of the 13th annual international cryptology conference on Advances in cryptology, New York, NY, USA, Springer-Verlag New York, Inc. (1994) 13–21
6. Bernstein, D.J., Buchmann, J., Dahmen, E.: Post Quantum Cryptography. Springer Publishing Company, Incorporated (2008)
7. Biswas, B., Sendrier, N.: McEliece Cryptosystem Implementation: Theory and Practice. In: PQCrypto. (2008) 47–62
8. Berger, T.P., Cayrel, P.L., Gaborit, P., Otmani, A.: Reducing Key Length of the McEliece Cryptosystem. In: AFRICACRYPT '09: Proceedings of the 2nd International Conference on Cryptology in Africa, Berlin, Heidelberg, Springer-Verlag (2009) 77–97
9. Berger, T.P., Loidreau, P.: How to Mask the Structure of Codes for a Cryptographic Use. Designs, Codes and Cryptography **35** (2005) 63–79 10.1007/s10623-003-6151-2.
10. Otmani, A., Tillich, J.P., Dallot, L.: Cryptanalysis of Two McEliece Cryptosystems Based on Quasi-Cyclic Codes. Mathematics in Computer Science **3** (2010) 129–140

11. Misoczki, R., Barreto, P.: Compact McEliece Keys from Goppa Codes. In Jacobson, M., Rijmen, V., Safavi-Naini, R., eds.: Selected Areas in Cryptography. Volume 5867 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg (2009) 376–392
12. Goppa, V.D.: A new class of linear correcting codes. Problems of Information Transmission **6** (1970) 207–212
13. Infineon Technologies AG: SLE76 Product Data Sheet `http://www.infineon.com/cms/de/product/channel.html?channel=db3a3043156fd57301161520ab8b1c4c`.
14. Infineon Technologies AG: SLE 66CLX360PE(M) Family Data Sheet `http://www.infineon.com/dgdl/SPI_SLE66CLX360PE_1106.pdf?folderId=db3a304412b407950112b408e8c90004&fileId=db3a304412b407950112b4099d6c030a&location=Search.SPI_SLE66CLX360PE_1106.pdf`.
15. Eisenbarth, T., Güneysu, T., Heyse, S., Paar, C.: MicroEliece: McEliece for Embedded Devices. In: CHES '09: Proceedings of the 11th International Workshop on Cryptographic Hardware and Embedded Systems, Berlin, Heidelberg, Springer-Verlag (2009) 49–64
16. Strenzke, F.: A Smart Card Implementation of the McEliece PKC. In: Information Security Theory and Practices. Security and Privacy of Pervasive Systems and Smart Devices. Volume 6033 of Lecture Notes in Computer Science., Springer Berlin / Heidelberg (2010) 47–59
17. Heyse, S.: Low-Reiter: Niederreiter Encryption Scheme for Embedded Microcontrollers. In Sendrier, N., ed.: Post-Quantum Cryptography. Volume 6061 of Lecture Notes in Computer Science., Springer Berlin / Heidelberg (2010) 165–181
18. German Federal Bureau of Information Security (BSI): Technical Guideline TR-03110: Advanced Security Mechanisms for Machine Readable Travel Documents, Version 2.02 (2009)
19. German Federal Bureau of Information Security (BSI): Technical Guideline TR-03111: Elliptic Curve Cryptography, Version 1.11 (2009)
20. Bernstein, D.J., Lange, T., Peters, C.: Attacking and defending the McEliece cryptosystem. Post-Quantum Cryptography, LNCS **5299** (2008) 31–46
21. Kobara, K., Imai, H.: Semantically secure McEliece public-key cryptosystems - conversions for McEliece PKC. Practice and Theory in Public Key Cryptography - PKC '01 Proceedings (2001)
22. Cooper et al.: RFC 5280 `http://tools.ietf.org/html/rfc5280`.
23. Coronado, L.C., Buchmann, J., Carlos, L., Garcia, C., Dahmen, E., Klintsevich, E., Darmstadt, T.U.: CMSS – An Improved Merkle Signature Scheme Johannes Buchmann (2006) `www.cdc.informatik.tu-darmstadt.de/~dahmen/papers/BCDDK06.pdf`.
24. Henry S. Warren: Hacker's Delight (2003)
25. National Institute of Standards and Technology: FIPS PUB 180-3 Secure Hash Standard
26. Olivier Gay: `http://www.ouah.org/ogay/sha2/`.
27. Witschnig, H., Patauner, C., Maier, A., Leitgeb, E., Rinner, D.: High speed RFID lab-scaled prototype at the frequency of 13.56 MHz. e & i Elektrotechnik und Informationstechnik **124** (2007) 376–383 10.1007/s00502-007-0485-9.
28. Kocher, P.: Differential Power Analysis. Advances in Cryptology-CRYPTO'99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings **1666** 388–397
29. Peters, C.: Information-Set Decoding for Linear Codes over $\mathbb{F}_q$. In: PQCrypto. (2010) 81–94

30. Cayrel, P.L., Gaborit, P., Prouff, E.: Secure Implementation of the Stern Authentication and Signature Schemes for Low-Resource Devices. In: CARDIS '08: Proceedings of the 8th IFIP WG 8.8/11.2 international conference on Smart Card Research and Advanced Applications, Berlin, Heidelberg, Springer-Verlag (2008) 191–205