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 through an example implementation and give arguments that it will be possible for a smart card controller to carry out the associated computations to sustain the transmission rates of possible future high speed contactless interfaces.

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 problem 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]. All recent proposals that reduce the key size using other codes than in the original McEliece scheme will have to prevail for some time until they can be granted the same trust as the original scheme, employing classical binary Goppa codes [11], the security of which is still unquestioned after 33 years.

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

Accordingly, in this work, we address the problem of performing the public operations, i.e. encryption or signature verification, of conventional code-based public key cryptosystems with large public keys 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 [12, 13]. 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 [14–16] 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 6. Next, in Section 7 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.

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 [17]. There, an elliptic curve based key agreement scheme is realized [18]. 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 [19] 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 [20]. 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 [21]. 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 [22] 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).

4 Transmission Rates

In this section, we give an overview of transmission rates available for embedded systems, espacially smart card microcontrollers.

For instance a SLE66CLX360PE [13] smart card platform from Infineon Technologies AG 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.

In the future, contactless transmission rates may be about 837.500 bytes/s [23], i.e. about 8 times higher than the rate considered above[3]. In the following section, we will show that 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. Note that in this case there are still about 35 cpu cycles available between the receival of two bytes.

5 Example Implementation

We implemented the proposed approach on an ATUC3A1512 32-bit microcontroller from Atmel's AVR32 family. We chose an embedded 32-bit platform ba-

[3] In the referenced work, this transmission rate is actually only achieved in the direction from the card to the reader. However, we want to use it merely as an orientation for the transmission rates achievable in the near future.

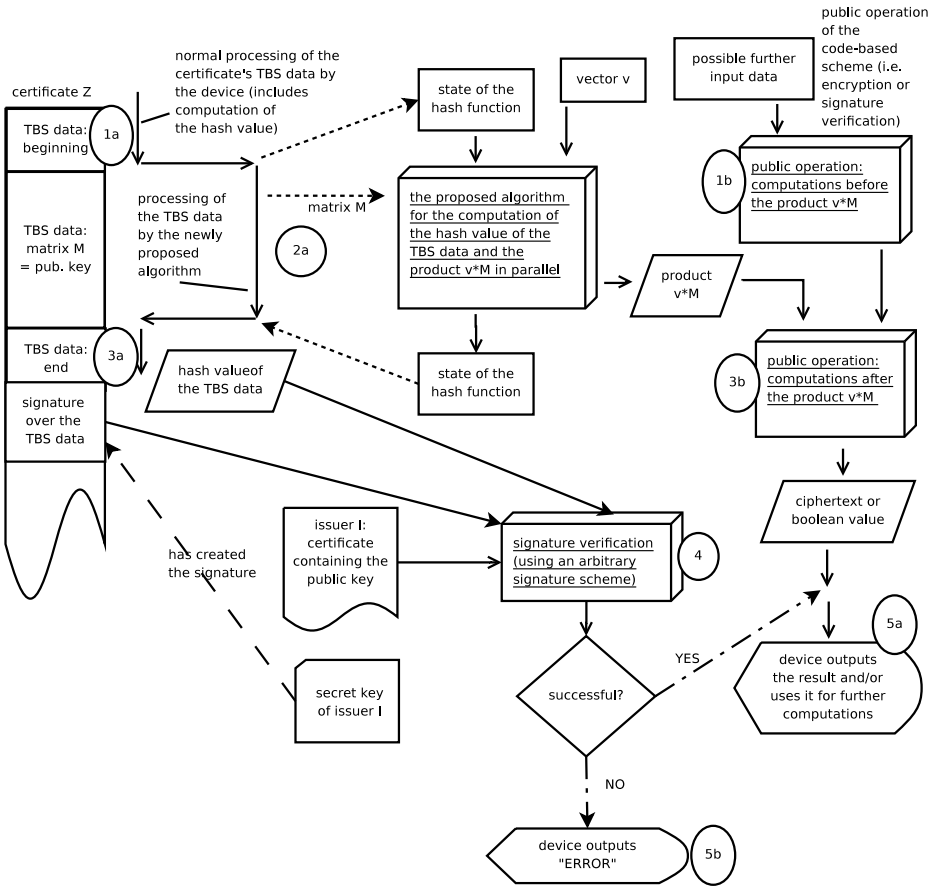

Fig. 1. Overview of the complete process of the online public operation.

sically because SHA256 is designed for 32-bit platforms. There also exist 32-bit smart card controllers [24], thus our evaluations are significant for this type of platform.

The personal computer (PC) communicates with the AVR32 over a serial line. For the implementation of the serial communication, on the AVR32, we used the API for the device's Universal Asynchronous Receiver Transmitter (UART) provided by Atmel. On the side of the personal computer, we used the API to the serial port of the Linux operating system. The PC can send commands to the AVR32, which are formed by a six byte header and optional payload data, the length of which is encoded the last four header bytes. The first header byte is zero for all commands, and the second byte determines one of the following commands:

- set the vector to multiply
- carry out the online multiplication (starts an interactive protocol for the matrix transmission described below)
- get the multiplication result from the AVR32
- get the hash result from the AVR32

The AVR32 responds to these commands by sending a two byte status code and optional data payload preceding the status code, or in the case of the online multiplication command, by starting an interactive protocol.

This protocol is depicted in Figure 2. As a precondition, the vector to multiply has to be set in the device through the corresponding command. After the receival of the online multiplication command (which does not carry payload data), the AVR32 sets up two buffers $B1$ and $B2$ which are of an equal predefined size. It sends a two byte acknowledgement (ACK) code to the PC as the answer to the command. Then the PC sends the first matrix part which is of equal size as the buffers $B1$ and $B2$. The receival of a single byte over the UART interface of the AVR32 triggers an interrupt which is serviced by an Interrupt Service Routine (ISR) which writes the byte to the next free position in $B1$. After the first block has been received completely, the AVR32 sends another ACK code to the PC, who in turn reacts by sending the next part. At this point the AVR32 exchanges the role of the buffers $B1$ and $B2$: the data is now received to $B2$ (which did not play any role while receiving the first part), and $B1$, containing the first matrix part, is fed into the SHA256 computation and the matrix multiplication. Both, the hashing and matrix multiplication are implemented as objects which can be updated by calling routines that take arbitrary amounts of data as an argument.

For hash functions, this is the standard implementation technique. Because demanded by our approach, we adopted this technique for the matrix multiplication. In our implementation, the matrix-vector multiplication is carried out column-wise. The advantages and disadvantages of this approach in contrast to row-wise multiplication is discussed in Section 6.1. The multiplication object knows the number of rows and columns of the matrix and has the source vector set. As the matrix data is fed column-wise it keeps track of the current row and column position. It processes the current column by carrying out the logical

AND (multiplication in $\mathbb{F}_2$) between the matrix column and the vector 32-bit word-wise, and computes the XOR (addition in $\mathbb{F}_2$) with a 32-bit accumulator. When a column is finished, the parity (i.e. sum of all the word's bits in $\mathbb{F}_2$) of the accumulator is written to the corresponding result bit.

5.1 Non-interactive Version of the Protocol

However, it turned out that the interactive protocol incurs significant delay in the communication which results from the fact that our PC program is running in userspace and thus sending and receiving data via the serial interface is delayed. If the protocol were implemented in a card terminal, which could be the case in a real world implementation of the online multiplication, or at the PC side in a kernel mode driver, such issues would not arise. To show the efficiency of the approach, we modified the protocol depicted in Figure 2: the AVR32 does not send any ACK answers beyond the very first one. Consequently, the matrix data is sent as a continuous stream after the AVR32 has sent the initial ACK. In this way, the protocol looses the feature that it works independently of the ratio of transmission speed and computation speed: in this non-interactive setting, it must be guaranteed that the hash and multiplication computation of the processed buffer has finished before the receive buffer has been completely filled. With this approach, the performance, could be improved by a factor of roughly 1.3. The concrete results are discussed shortly.

5.2 Simulation of higher Transmission Rates

On the chosen AVR32 platform, the maximal transmission speed is given by a baudrate of 460,800. In the RS232 transmission format each data byte is encoded in 10 bits, yielding a net transmission rate of 46,080 byte/s. In order to demonstrate the computation speed that would be possible beyond this limitation, we implemented a means of simulating higher transmission speeds. This is achieved by creating a matrix whose rows have repetitive entries, i.e. the values of 8-bit chunks repeats r times. An example of the beginning of a row for $r = 4$ would be

```
0x1D, 0x1D, 0x1D, 0x1D, 0xA3, 0xA3, 0xA3, 0xA3, 0x22, ...
```

In this setting, on the PC side such a repetitive matrix is generated. When the matrix is transmitted, however, each repeated element is sent only once. On the receiving side, the repetition value r is also known and each received byte is appended to the buffer r times. In this way, we simulate a transmission rate $B_{\text{sim}} = rB_{\text{real}}$, where B_{real} is the actual UART transmission rate.

Table 5.2 shows the measurement results for interactive version of the protocol as depicted in Figure 2, and the non-interactive version described in the previous paragraph in the two left columns. Here, we used a matrix with 1000 rows and 800 columns, i.e. yielding a size of 100,000 bytes. This size in the domain of McEliece public keys with 100 bit security [15]. The reason for this specific

choice in contrast to using the exact parameters resulting from a real code is to simplify the implementation, for instance choosing the using a row length divisible by 8 avoids the necessity to implement the special treatment of only partially used final bytes. In all our measurements the CPU speed of the AVR32 was set to 33MHz, since also todays contactless smart card platforms run at approximately this speed [12]. The referenced platform only runs at 30MHz, using this in our implementation showed that at this CPU speed the (simulated) transmission rate given in the rightmost column could not be supported in the experiment. The transmission speed of 386,640 bytes/s is only approximately half of that of the research implementation presented in [23] already mentioned in Section 4. Thus our results show that even without dedicated hardware, todays embedded platforms already enable computation speeds for the hash computation and matrix multiplication in the domain of the associated transmission rates that can be exptected to be supported by contactless devices in the near future. This makes it feasible that with adequate hardware support the full 837.500 Byte/s rate given in [23] can be supported by the throughput of the computational tasks.

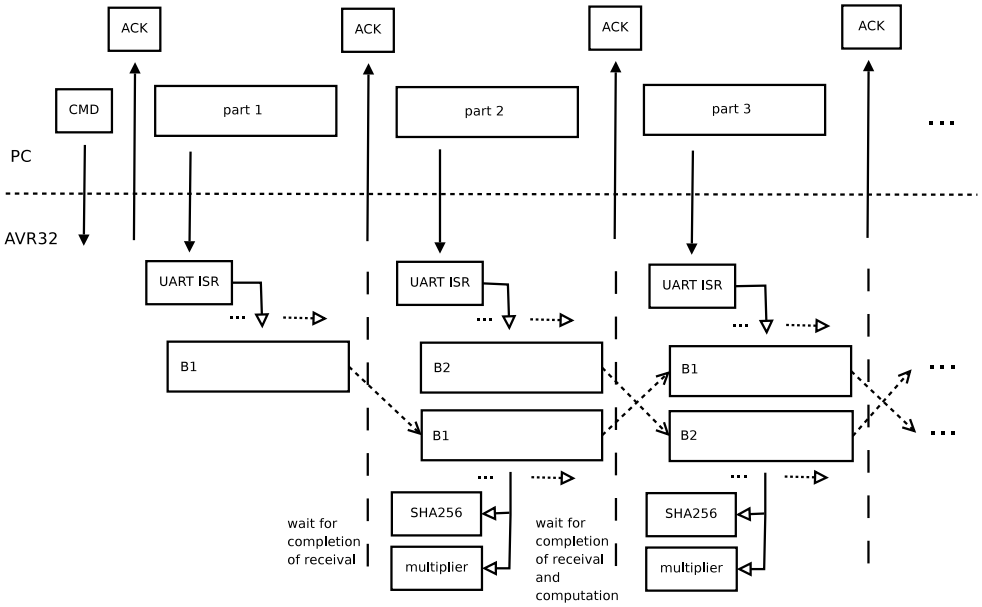

Fig. 2. Schematic overview of the interrupt based implementation of the online multiplication.

The hash implementation is based on the open source implementation [25]. The C source code allows for complete unrolling of the SHA256 compression

	based on computation throughput	experimental result - w/o ACK
cycles/byte	measured: 55.6 for SHA256, 4.2 for mult. yields: **59.8**	92
time at 33MHz CPU for 100,000 Bytes	181ms	**279ms**
transmission rate in bytes/s	551.839	$B_{\text{sim}} = 368,640$ ($r = 8$)

Table 1. Performance of the SHA256 and binary matrix multiplication on the AT32UC3A1 platform. The results on the first column are based on the throughput benchmarking results for the two computational tasks when processing of buffers of 8192 bytes. Both the time for the online multiplication and the transmission rate necessary to support the throughput of both computational tasks are theoretically derived from the former. In the second column, the time of the whole online matrix multiplication with the given transmission rate $B_{\text{sim}} = 8 \cdot 46,080$ byte/s was measured on the ATUC3A1512 platform. Here the a receive buffer size of of 1536 bytes was used.

function through a macro definition. Activating loop unrolling resulted in a performance gain of 1.6 for the hash function computation. All further performance data is based on this implementation choice.

6 Variants of the proposed Approach

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

The 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 about half of vector a's bits will have value zero. 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 [26], 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.

7 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.

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 [16], 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. Goppa, V.D.: A new class of linear correcting codes. Problems of Information Transmission **6** (1970) 207–212
12. Infineon Technologies AG: SLE76 Product Data Sheet `http://www.infineon.com/cms/de/product/channel.html?channel=db3a3043156fd57301161520ab8b1c4c`.
13. 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`.
14. 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
15. 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
16. 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
17. German Federal Bureau of Information Security (BSI): Technical Guideline TR-03110: Advanced Security Mechanisms for Machine Readable Travel Documents, Version 2.02 (2009)

18. German Federal Bureau of Information Security (BSI): Technical Guideline TR-03111: Elliptic Curve Cryptography, Version 1.11 (2009)
19. Bernstein, D.J., Lange, T., Peters, C.: Attacking and defending the McEliece cryptosystem. Post-Quantum Cryptography, LNCS **5299** (2008) 31–46
20. 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)
21. Cooper et al.: RFC 5280 `http://tools.ietf.org/html/rfc5280`.
22. 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`.
23. 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.
24. Infineon Technologies AG: SLE78 Product Data Sheet `http://www.infineon.com/cms/en/product/channel.html?channel=db3a30431ce5fb52011d47b166342af0`.
25. Olivier Gay: `http://www.ouah.org/ogay/sha2/`.
26. 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