Towards One Cycle per Bit Asymmetric Encryption: Code-Based Cryptography on Reconfigurable Hardware

Stefan Heyse and Tim Güneysu

Horst Görtz Institute for IT-Security
Ruhr-Universität Bochum
44780 Bochum, Germany
{heyse,guneysu}@crypto.rub.de

Abstract. Most advanced security systems rely on public-key schemes based either on the factorization or the discrete logarithm problem. Since both problems are known to be closely related, a major breakthrough in cryptanalysis tackling one of those problems could render a large set of cryptosystems completely useless.

Code-based public-key schemes are based on the alternative security assumption that decoding generic linear binary codes is NP-complete. In the past, most researchers focused on the McEliece cryptosystem, neglecting the fact that the scheme by Niederreiter has some important advantages. Smaller keys, more practical plain and ciphertext sizes and less computations. In this work we describe a novel FPGA implementation of the Niederreiter scheme, showing that its advantages can result a very efficient design for an asymmetric cryptosystem that can encrypt more than 1.5 million plaintexts per seconds on a Xilinx Virtex-6 FPGA, outperforming all other popular public key cryptosystems by far.

1 Introduction

Public-key cryptosystems build the foundation for virtually all advanced cryptographic requirements, such as asymmetric encryption, key exchange and digital signatures. However, up to now most cryptosystems rely on two classes of fundamental problems to establish security, namely the factoring problem and the (elliptic curve) discrete logarithm problem. Since both are related, a significant cryptanalytical improvement will turn out a large number of currently employed security systems to be insecure overnight. This threat is further nourished by upcoming generations of powerful quantum computers that have been shown to be very effective in computing solutions to the problems mentioned above [36]. Recently, IBM announced two further breakthrough in quantum computing [9] so that such practical systems might already become available in the next 15 years. Evidently, a larger *diversification* of cryptographic primitives that are resistant against such attacks is absolutely essential for the future of public-key cryptosystems.

Addressing this, cryptosystems that settle their security on alternative hard problems have gathered much attention in the last years, such as multivariate-quadratic ($\mathcal{MQ}$ -), lattice-based and code-based schemes. A drawback of these constructions have been their low efficiency, large key sizes or complex computation with respect to RSA and ECC, what typically makes it difficult to employ them on small and embedded systems. First approaches to tackle these issues on such small systems has been presented for $\mathcal{MQ}$ and McEliece cryptosystems on the last years' workshops of the CHES series [8,14].

In this work, we focus on another code-based scheme. The cryptosystem presented by Niederreiter [29] is dual to McEliece's proposal [25] but enables higher efficiency while still maintaining the same security argument. In particular, it has been shown that even after more than 30 years of thorough analysis the code-based schemes remain unbroken when security parameters and fundamental codes are appropriately chosen [5]. Furthermore, a recent result indicated that McEliece and Niederreiter cryptosystems also resist quantum computing [12].

Our Contribution: In this work, we present the first implementation of the Niederreiter scheme on reconfigurable hardware. Our implementation for Xilinx' Virtex-6 FPGAs provides 80-bit of equivalent symmetric security and can run more than 1.5 million encryption and 17000 decryption operations per second, respectively. By using only a moderate amount of memory and logic resources, our implementation even outperforms many other implementation of classical cryptosystems, such as ECC-160 and RSA-1024. This impressive throughput of our implementation has become possible due to our highly optimized constant weight encoding algorithm. Due to these optimizations and the inherent advantages of Niederreiter over McEliece, we achieve a performance that are even orders of magnitudes faster than any other McEliece implementation reported so far.

Outline: This paper is structured as follows: we start with a brief introduction to Niederreiter encryption, shortly explain necessary operations on Goppa codes and introduce constant weight encoding. Section 4 describe our actual implementations for an Xilinx Virtex6LX240 FPGA. Finally, we present our results for these platforms in Section 5.

2 Previous Work

Although proposed already more than 30 years ago, the code-based encryption such as the McEliece and Niederreiter scheme has never gained much attention due to their large secret and public keys involved. With their large and costly memory requirements they have been hardly integrated in any real-world products – yet. The first FPGA-based implementation of McEliece was proposed in [14] for a Xilinx Spartan-3AN and encrypts and decrypts data in 1.07 *ms* and 2.88 *ms*, using security parameters achieving an equivalence of 80-bit symmetric security. The authors of [38] presented another accelerator for binary McEliece encryption on a more powerful Virtex5-LX110T that encrypts and decrypts in 0.5 *ms* and 1.4 *ms* providing a similar level of security. For x86 personal computers, the most recent implementation of the binary McEliece scheme is due

to Biswas and Sendrier [7] that achieves about 83-bit security according to [5]. Comparing their implementation with other public key schemes, it turns out that McEliece encryption is even faster than RSA and NTRU [4] - at the cost of larger keys. But researchers addressed the issue of large keys by replacing the original used binary Goppa codes with codes that allow more compact representations, e.g., [26,32]. However, for these schemes only PC and microcontroller implementations exist so far [2,20]. Even worse, most of the attempts to reduce the key size have been broken[15]. Note that all previous works also exclusively target the McEliece cryptosystems. To the best of our knowledge, the only published Niederreiter implementation for embedded systems is an implementation for small 8-bit AVR microcontrollers enabling encryption and decryption in 1.6 *ms* and 179 *ms* [19]. Additionally, there are some Java based implementations or Niederreiter based signatures available [33].

2.1 The Niederreiter Public Key Scheme

The Niederreiter scheme [29] is a public key cryptosystem based on linear error-correcting codes, similar to the popular McEliece cryptosystem. The secret key is an efficient decoding algorithm for an error-correcting code with dimension k , length n and error correcting capability t . To create a public key, Niederreiter defined a random $n \times n$ -dimensional permutation matrix P disguising the structure of the code by computing the product $\hat{H} = M \times H \times P$. Here, M can be chosen as the $(n - k) \times (n - k)$ matrix that transforms $\hat{H}$ into systematic form. Using the public key $K_{pub} = (\hat{H}, t)$ and private key $K_{sec} = (P^{-1}, H, M^{-1})$, encryption and decryption can be defined as in Algorithm 1 and Algorithm 2, respectively.

Algorithm 1. Classical Niederreiter Message Encryption

Input: Message m , $K_{pub} = (\hat{H}, t)$

Output: Ciphertext c

- 1: Encode the message m as a binary string of length n and weight t called e
 - 2: $c = \hat{H}e^T$
 - 3: **return** c
-

Note that Algorithm 1 employs only a simple matrix multiplication operation on the input message that was transformed into a *constant weight word* before. The necessary algorithm for constant weight encoding (Bin2CW) is given in detail in Section 2.4.

Algorithm 2. Classical Niederreiter Message Decryption

Input: Ciphertext c , $K_{sec} = (P, M, g(z), \mathcal{L})$

Output: Message m

- 1: $c' \leftarrow M^{-1}c$
 - 2: decode c' to error vector $e' = Pe^T$
 - 3: $e \leftarrow P^{-1}e'$
 - 4: Decode the error vector e to the binary message m
 - 5: **return** m
-

Decryption is the most time-consuming process and requires several complex operations on the linear code defined over binary extension fields. In Section 2.5 we briefly introduce the required steps for decoding codewords. Note that in a modern Niederreiter description, applying the permutation P can be omitted completely. This is possible by computing a permuted support and merging the mapping of roots to bit positions via the support list with the permutation. See Section 4.2 for details.

2.2 Niederreiter vs. McEliece

The main difference between Niederreiter and McEliece is the public key. While an $(n \times k)$ generator matrix serves as public key in McEliece, Niederreiter uses a $(n \times (n - k))$ parity check matrix for this purpose. Both matrices can be used in their systematic form, leading to $((n - k) * k)$ bits storage requirement in both cases. Using this method in the McEliece case demands a CCA2 secure conversion [30,24] to stay secure, whereas Niederreiter can be used without this overhead. In McEliece, an n -bit code word with errors is used as ciphertext, whereas Niederreiter uses the $(n - k)$ -bit syndrome as plaintext. This shifts the syndrome computation from the receiver to the sender of the message and therefore speeds up decryption, still maintaining high encryption performance. At the same time, the parity check matrix and related information is no longer part of the secret key, thus reducing the secret key size. However, the Niederreiter scheme requires the scrambling matrix M in any case which can be omitted when using McEliece encryption. Finally, Niederreiter encryption imposes less restrictions on the plaintext size, i.e., depending on the parameter sets and constant weight encoding algorithm, Niederreiter enables plaintext blocks with a size of only hundreds of bits instead of several thousands bits as in the case of McEliece encryption. In particular, for key transportation protocols with symmetric key sizes of 128 to 256 bits, the transfer of thousands of bits as required in the case of McEliece can be an expensive overhead.

2.3 Security Parameters

All security parameters for cryptosystems are chosen in a way to provide sufficient protection against the best known attack (whereas the notion of “sufficient” is determined by the requirements of an application). On the attempt to employ an alternative cryptosystem, it is of utmost important for a security engineer to being able to safely assess if this best attack has already been found. In this context, the work by Bernstein *et al.* [5] currently proposes the best attack on McEliece and Niederreiter cryptosystems so far reducing the work factor to break the McEliece scheme based on a $(1024, 524)$ Goppa code and $t = 50$ to $2^{60.55}$ bit operations. According to their findings, we summarize the security parameters for specific security levels in Table 1. The public key size column gives the size

Table 1. Security Parameters for Niederreiter Cryptosystems

Security Level	Parameters	Size K_{pub}	Size K_{sec}
	(n, k, t) , errors added	in KBits	$(g(z) \mid \mathcal{L} \mid M^{-1})$ KBits
Short-term (60 bit)	(1024, 644, 38), 38	239	(0.37 10 141)
Mid-term I (80 bit)	(2048, 1751, 27), 27	507	(0.29 22 86)
Mid-term II (128 bit)	(2690, 2280, 56), 57	913	(0.38 18 164)
Long-term (256 bit)	(6624, 5129, 115), 117	7,488	(1.45 84 2,183)

of a systematic parity check matrix and the secret key column the size of the Goppa polynomial $g(z)$, the support $\mathcal{L}$ and the inverse scrambling matrix M^{-1} .

As can be clearly seen, the main caveat against coding based cryptosystems is the significant size of the public and private keys. For 80 bit security, for example, the parameters $m = 11, n = 2048, t = 27, k \geq 1751$ already lead to an M^{-1} of 11 KBytes and a public key size of 63 KBytes. Note that we can reduce the size of the public key from originally 74 KBytes by choosing M in such a way that it brings $\hat{H}$ to systematic form $\hat{H} = (ID_{n-k} \mid Q)$, where only the redundant part Q has to be stored.

2.4 Constant Weight Encoding and Decoding

Before encrypting a message with the Niederreiter cryptosystem, the message has to be encoded into an error vector. More precisely, the message needs to be transformed into a bit vector of length n and constant weight t . There exist quite a few encoding algorithms (e.g., those in [11,34,16]), however they are not directly applicable to the restricted execution environment of embedded systems and hardware. In this work we unfolded the recursive algorithm proposed in [35] so that it can run by an iterative state machine. During the encoding operation, one has to compute a value $d \approx \frac{\ln(2)}{t} \cdot (n - \frac{t-1}{t})$ to determine how many bits of the message are encoded into the distance to the next one-bit on the error vector. But many embedded (hardware) systems do not have a dedicated floating-point and division unit so these operations should be replaced. We therefore substituted the floating point operation and division by a simple and fast table lookup. Since we still preserve all properties from [35], the algorithm will still terminate with a minor negligible loss in efficiency.

The encoding algorithm suitable for embedded systems is given in Algorithm 3.

The constant weight decoding algorithm was adapted in a similar way, and is presented in Algorithm 4.

2.5 Encoding and Decoding of Goppa Codes

In this section we briefly introduce the underlying Goppa codes that provide the fundamental arithmetic for the Niederreiter cryptosystem.

Theorem 1. Let $g(z)$ be an irreducible polynomial of degree t over $GF(2^m)$. Then the set

$$\Gamma(G(z), GF(2^m)) = \{(c_\alpha)_{\alpha \in GF(2^m)} \in \{0, 1\}^n \mid \sum_{\alpha \in GF(2^m)} \frac{c_\alpha}{z - \alpha} \equiv 0 \bmod G(z)\} \quad (1)$$

defines a binary Goppa code C of length $n = 2^m$, dimension $k \geq n - mt$ and minimum distance $d \geq 2t + 1$. The set of the α_i is called the support $\mathcal{L}$ of the code.[42]

Algorithm 3. Encode Binary String in Constant Weight Word (Bin2CW)

Input: n, t , binary stream B

Output: $\Delta[0, \dots, t-1]$

```

1:  $\delta = 0, index = 0$ 
2: while  $t \neq 0$  do
3:   if  $n \leq t$  then
4:      $\Delta[index++] = \delta$ 
5:      $n- = 1, t- = 1, \delta = 0$ 
6:   end if
7:    $u \leftarrow uTable[n, t]$ 
8:    $d \leftarrow (1 << u)$ 
9:   if  $read(B, 1) = 1$  then
10:     $n- = d, \delta+ = d$ 
11:   else
12:     $i \leftarrow read(B, u)$ 
13:     $\Delta[index++] = \delta + i$ 
14:     $\delta = 0, t- = 1, n- = (i + 1)$ 
15:   end if
16: end while
```

Algorithm 4. Decode Constant Weight Word to Binary String (CW2Bin)

Input: $n, t, \Delta[0, \dots, t-1]$

Output: binary stream B

```

1:  $\delta = 0, index = 0$ 
2: while  $t \neq 0$  AND  $n > t$  do
3:    $u \leftarrow uTable[n, t]$ 
4:    $d \leftarrow (1 << u)$ 
5:   if  $\Delta[index] \geq d$  then
6:      $Write(1, B)$ 
7:      $\Delta[index]- = d$ 
8:      $n- = d$ 
9:   else
10:     $\delta = \Delta[index++]$ 
11:     $Write(0|\delta, B)$ 
12:     $n- = (\delta + 1), t- = 1$ 
13:   end if
14: end while
```

There exist fast decoding algorithms with runtime $O(n \cdot t)$ operations (e.g., [31,41]). For each irreducible polynomial $g(z)$ over $GF(2^m)$ of degree t there exists a binary Goppa code of length $n = 2^m$ and dimension $k = n - mt$. This code is capable to correct up to t errors [3] and can be represented by a $k \times n$ generator matrix G such that $C = \{mG : m \in F_2^k\}$.

Since $r = c + e \equiv e \pmod{g(z)}$ holds, the syndrome $Syn(z)$ of a received codeword can be obtained from Equation (1) by

$$Syn(z) = \sum_{\alpha \in GF(2^m)} \frac{r_\alpha}{z - \alpha} \equiv \sum_{\alpha \in GF(2^m)} \frac{e_\alpha}{z - \alpha} \pmod{G(z)} \quad (2)$$

To finally recover $\underline{e}$, we need to solve the key equation $\sigma(z) \cdot Syn(z) \equiv \omega(z) \pmod{g(z)}$, where $\sigma(z)$ denotes a corresponding error-locator polynomial and $\omega(z)$ denotes an error-weight polynomial. Note that it can be shown that $\omega(z) = \sigma(z)'$ is the formal derivative of the error-locator and by splitting $\sigma(z)$ into even and odd polynomial parts $\sigma(z) = a(z)^2 + z \cdot b(z)^2$, we finally determine the following equation to determine error positions:

$$Syn(z)(a(z)^2 + z \cdot b(z)^2) \equiv b(z)^2 \pmod{G(z)} \quad (3)$$

To solve Equation (3) for a given syndrome $Syn(z)$, the following steps have to be performed:

1. Compute an inverse polynomial $T(z)$ with $T(z) \cdot Syn(z) \equiv 1 \pmod{g(z)}$ (or provide a corresponding table). It follows that $(T(z) + z)b(z)^2 \equiv a(z)^2 \pmod{g(z)}$.
2. There is a simple case if $T(z) = z \Rightarrow a(z) = 0$ s.t. $b(z)^2 \equiv z \cdot b(z)^2 \cdot Syn(z) \pmod{G(z)} \Rightarrow 1 \equiv z \cdot Syn(z) \pmod{G(z)}$ which directly leads to $\sigma(z) = z$. Contrary, if $T(z) \neq z$, compute a square root $R(z)$ for the given polynomial $R(z)^2 \equiv T(z) + z \pmod{G(z)}$. Based on an observation by Huber [23] this can be done by a simple polynomial multiplication. We can then determine solutions $a(z), b(z)$ satisfying

$$a(z) = b(z) \cdot R(z) \pmod{G(z)}. \quad (4)$$

using the extended euclidean algorithm. The computation is stopped, when $a(z)$ reaches degree $\lfloor \frac{t}{2} \rfloor$. Finally, we use the identified $a(z), b(z)$ to construct the error-locator polynomial $\sigma(z) = a(z)^2 + z \cdot b(z)^2$.

3. The roots of $\sigma(z)$ denote the positions of error bits.

Searching the roots of $\sigma(z)$ with degree t over $GF(2^m)$ is time-consuming. Besides the plain evaluation of all support elements, for this two most commonly used methods are the Chien search [10] and Horner's scheme [22]. A third method as proposed in [6] for PC platforms can not be easily parallelized and requires a lot of greatest common divisor and trace computations. In our work we therefore use a parallelized version of the Chien search, which concurrently evaluates all t coefficients (see Section 4.2 for details). The decoding process, as required in Step 2 of Algorithm 2 for message decryption, is finally summarized in Algorithm 5.

Algorithm 5. Decoding Goppa Codes

Input: Received syndrome $\text{Syn}(z)$ of a codeword r with up to t errors**Output:** Recovered message $\hat{m}$

```

1:  $T(z) \leftarrow \text{Syn}(z)^{-1} \bmod G(z)$ 
2: if  $T(z) = z$  then
3:    $\sigma(z) \leftarrow z$ 
4: else
5:    $R(z) \leftarrow \sqrt{T(z) + z}$ 
6:   Compute  $a(z)$  and  $b(z)$  with  $a(z) \equiv b(z) \cdot R(z) \bmod G(z)$ 
7:    $\sigma(z) \leftarrow a(z)^2 + z \cdot b(z)^2$ 
8: end if
9: Determine roots of  $\sigma(z)$  and compute  $e$ 
10: return  $e$ 

```

3 Design Decisions

In this section we discuss the design and parameter decisions for our Niederreiter implementation on reconfigurable hardware. A primary goal of our design is high-performance, a secondary reasonable hardware costs.

3.1 Parameter Selection

With the implementation of our Niederreiter cryptosystem, we aim to provide 80-bit of equivalent symmetric security, i.e., protection that is comparable to the security of ECC and RSA with approximately 160-bit and 1024-bit, respectively. This level of security is still considered sufficient for mid-term security applications providing a reasonable cost-performance ratio and thus suitable for most embedded systems. To achieve this level of security, we selected the parameters $m = 11, n = 2048, t = 27, k \geq 1751$ resulting in a private and public key size of 13.5 and 63 KBytes to be stored on the device. This amount of memory is available in each Xilinx FPGA larger than the low-cost Spartan-3 XC3S1000, Virtex-5 XC5VLX30 and Virtex-6 XC6VLX75T, respectively [44].

The above security level was originally proposed to minimize public key size for a given security level and not to maximize performance. However, we stay with this parameters to be comparable with the existing code-based implementations, which all selected this parameter set.

For practical purposes, we fixed the size of a message block to 192 bits which can be encoded into an appropriate error vector in any case. Note that the constant weight encoding algorithm requires an input of variable length to produce a constant weight output. Experiments showed that on average 210 bits are required to construct a valid constant weight word. Fixing the input message to 192 bits and adding random bits as required, makes the algorithm practicable without leaking any security-relevant information.

The secret key consisting of the Goppa polynomial $g(z)$, the support $\mathcal{L}$ and the inverse scrambling matrix M^{-1} is stored as part of the bitstream file which

configures the FPGA. Because only the Spartan3-AN class from Xilinx offers internal Flash memory to store the bitstream internally in a (somewhat) protected way, appropriate actions have to be taken to protect the bitstream when storing it in external memory. In this case it is mandatory to enable bitstream encryption using AES-256 which is available for larger Xilinx Spartan-6 and all Xilinx Virtex-FPGAs from Virtex-4. Note, however, that also the Xilinx specific bitstream encryption [43] was successfully attacked by side-channel analysis in [28].

The (larger) public key can be stored either in internal or external memory since it does not require special protection. For our implementation we opted to store the public key in internal BRAMs to allow immediate access for high-performance encryption.

3.2 Inherent Side Channel Resistant

Some research had been done regarding side channels in code-based cryptography, however, all solely focused on implementations of McEliece encryption [40,39,37,21,27]. The advantage of Niederreiter in contrast to McEliece is that the ciphertext not consists only out of a pure codeword with randomly added errors. Fault attacks cannot be easily performed by flipping random bits of the ciphertext assuming that the decoder either corrects one of the the intentionally injected errors or fails to do so. For Niederreiter encryption, the ciphertext is a syndrome polynomial. Flipping random bits will result most likely in a decoding error without leaking any information. This renders all the attacks from [40,39,37] useless. Only power analysis attacks, like the one described in [21], which directly attacks the Goppa polynomial used in the Patterson algorithm, are still possible. It also requires further investigation, if adoptions of attacks targeting the root search are possible.

4 Implementation

This section describes our implementation primarily targeting a recent Virtex-6 LX240 FPGA. Note that this device is certainly too large for our implementation but was chosen due to its availability on the Xilinx Virtex-6 FPGA ML605 Evaluation Kit for practical testing. Furthermore, we provide implementations for a Xilinx Spartan-3 and Xilinx Virtex-5 to allow fair comparisons with other work (cf. Table 4).

4.1 Encryption

The public key $\hat{H}$ is stored in an internal BRAM memory block and row-wise addressed by the output of the constant weight encoder. Multiplying a binary vector with a binary matrix is equivalent to a XOR operation of each row with input vector bit equal to one. Since this operation is trivial, we now focus on

the implementation of the constant weight encoding algorithm. Input data to our cryptosystem is passed using a FIFO with a non-symmetric 8-to-1 bit aspect ratio. Hence, after a word with 8-bit length is written to the FIFO, it can be read out bit by bit. This is the equivalent to the binary stream reader presented in Algorithm 3. Its main part is implemented as a small finite state machine. Every time a valid $\Delta[i]$ has been computed, it is directly transferred to the vector-matrix-multiplier summing up the selected rows. By interleaving operations we are able to process one bit from the FIFO at *every* clock cycle. After the last $\Delta[t]$ has been computed, only the last indexed row of $\hat{H}$ has to be added to the sum. Directly afterwards the encryption operation has finished and the ciphertext becomes available. Due to the very regular structure of the vector-matrix-multiplier and the small operands of the constant weight encoder, we were able to achieve a high clock frequency of 300 MHz. Nevertheless, the logic in the constant weight encoder is still the bottleneck.

Fig. 1. Block diagram of the encryption process

4.2 Decryption

The first step in the decryption process is the multiplication by the inverse matrix M^{-1} . This 11 KByte large matrix is stored in an internal BRAM and addressed by an incrementing counter. Using this BRAM, the rows of the matrix are XORed into an intermediate register if the corresponding input bit of the ciphertext equals to one. After $(n - k) = 297$ clock cycles, this register contains the value $c' = M^{-1} * c$ as shown in Algorithm 2. Now c' is passed on to the Goppa decoder. The Goppa decoder implements standard Patterson decoding (see Algorithm 5) and returns the erroneous bit positions in the order they are found during the decoding process.

Searching roots is a time-consuming process that is highlighted by Figure 2 showing our Chien search core. Decryption performance can be boosted by instantiating two or more of these cores in parallel and let them evaluate different support elements in parallel. Beside the additional management overhead in the controlling state machine, each of these cores requires additional 620 registers and 106 LUTs. We therefore only use one core which evaluates one support element in two clock cycles and finishes the entire process after 4098 clock cycles.

Fig. 2. Design of the Core for Chien Search

Next each root needs to be mapped to these bit positions. In this context, we constructed a table containing entries for $\mathcal{L} * P$, i.e., we merged this mapping and the reverse permutation from line 3 in Algorithm 2. As a side effect, the permutation P completely disappears from the scheme. Because the subsequent constant weight decoding algorithm expects the distance between the error bits in ascending order, we implemented a sorting circuit which sorts the error positions using a systolic implementation of bubble sort. At the same time it further computes the distance between two successive error positions. Finally, the error distances are translated into the binary message by a straightforward implementation of Algorithm 4 as developed in Section 4.1.

Fig. 3. Block diagram of the decryption process

5 Results

We now present the results for our implementation on three different platforms to enable a fair comparison with other work. Note that most of the differences in the number of used resources are due to architecture differences in the FPGA types, i.e., 4-input LUTs vs. 6-input LUTs and 18 KB BRAMs vs. 36 KB BRAMs in Spartan-3 and Virtex-5/6 FPGAs, respectively.

Encryption takes approximately 200 cycles and is around a factor of 72 faster than decryption that requires around 14,500 cycles. An open problem is how to transfer 1.5 million keys per second to the device, when many different public key are required, which is a typical application. Here an interface capable of transferring $1,5 \cdot 10^6 \cdot 63Kbyte \approx 774 \frac{Gbyte}{sec}$ would be necessary. As mentioned

Table 2. Implementation results of Niederreiter encryption with $n = 2048, k = 1751, t = 27$ after PAR

Aspect	Spartan3-2000	Virtex5-LX50	Virtex6-LX240
Slices	854 (2%)	291 (4%)	315 (1 %)
LUTs	1252 (3%)	888 (3%)	926 (1 %)
FFs	869 (2%)	930 (3%)	875 (1 %)
BRAMs	36 (90%)	18 (30%)	17 (4 %)
Frequency	150 MHz	250 MHz	300 MHz
CW Encode $e = encode(m)$	≈ 200 cycles		
Encrypt $c = e \cdot \hat{H}$	concurrently with CW Encoding		

Table 3. Implementation results of Niederreiter decryption with $n = 2048, k = 1751, t = 27$ after PAR

Aspect	Spartan3-2000	Virtex5-LX50	Virtex6-LX240
Slices	11253 (54%)	4077 (56%)	3887 (10 %)
LUTs	15559 (37%)	9743 (33%)	9409 (6 %)
FFs	13608 (33%)	13537 (47%)	12861 (4 %)
BRAMs	22 (55%)	13 (21%)	9 (2 %)
Frequency	95 MHz	180 MHz	250 MHz
Undo Scrambling $c \cdot M^{-1}$		297 cycles	
Compute $T = \text{Syn}(z)^{-1}$		4310 cycles	
Solve Equation (4)		4854 cycles	
Search Roots		4098 cycles	
Sort&Convert		85 cycles	
CW Decode		198 cycles	

Table 4. Comparison of our Niederreiter designs with single-core ECC and RSA implementations for 80 bit security

Scheme	Platform	Freq	Time/Op	Cycles/byte
This work [enc]	Virtex6-LX240T	300 MHz	0.66 μ s	8.3
This work [dec]	Virtex6-LX240T	250 MHz	58.78 μ s	612
McEliece [enc] [14]	Spartan3-AN1400	150 MHz	1070 μ s	768
McEliece [dec] [14]	Spartan3-AN1400	85 MHz	21,610 μ s	8788
This work enc	Spartan3-2000	150 MHz	1.32 μ s	8.3
This work dec	Spartan3-2000	95 MHz	154 μ s	612
McEliece [enc] [38]	Virtex5-LX110T	163 MHz	500 μ s	389
McEliece [dec] [38]	Virtex5-LX110T	163 MHz	1400 μ s	1091
This work [enc]	Virtex5-LX50T	250 MHz	0.793 μ s	8.2
This work [dec]	Virtex5-LX50T	180 MHz	81 μ s	612
ECC-P160 [17]	Spartan-3 1000-4	40 MHz	5.1 ms	10,200
ECC-K163 [17]	Virtex-II	128 MHz	35.75 μ s	224.6
RSA-1024 random [18]	Spartan-3A	133 MHz	48.54 ms	50,436
RSA-1024 random [18]	Spartan-6	187 MHz	34.48 ms	50,373
RSA-1024 random [18]	Virtex-6	339 MHz	19.01 ms	59,258
NTRU encryption [1]	Virtex 1000EFG860	50 MHz	5 μ s	8.3

above, the public-key cryptosystems RSA-1024 and ECC-P160 are assumed¹ to roughly achieve an similar level of 80 bit symmetric security [13]. We finally compare our results to published implementations of these systems that target

¹ According to [13], RSA-1248 actually corresponds to 80 bit symmetric security. However, no implementation results for embedded systems are available for this key size.

similar platforms (i.e., [14,38,17,18,1]). For a fair comparison with other existing implementations of code-based systems we also implemented our code for Spartan-3 and Virtex-5 FPGAs.

6 Conclusions

In this paper, we clearly demonstrate the very high performance that can be achieved with an efficient FPGA-based implementation of the Niederreiter scheme. Besides the more practical plaintext size and smaller public keys, the extremely high performance with more than 1.5 million encryption and 17000 decryption operations per second, respectively, makes Niederreiter an interesting candidate for applications where high throughput and many public key encryptions per second are required (e.g., upcoming car-2-car communication infrastructures).

Acknowledgements. The work described in this paper has been supported in part by the European Commission through the ICT programme under contract ICT-2007-216676 ECRYPT II. This work has been also been supported in part by the Ministry of Economic Affairs and Energy of the State of North Rhine-Westphalia (Grant 315-43-02/2-005-WFBO-009).

References

1. Bailey, D.V., Coffin, D., Elbirt, A., Silverman, J.H., Woodbury, A.D.: NTRU in Constrained Devices. In: Koç, Ç.K., Naccache, D., Paar, C. (eds.) CHES 2001. LNCS, vol. 2162, pp. 262–272. Springer, Heidelberg (2001)
2. Berger, T.P., Cayrel, P.-L., Gaborit, P., Otmani, A.: Reducing Key Length of the McEliece Cryptosystem. In: Preneel, B. (ed.) AFRICACRYPT 2009. LNCS, vol. 5580, pp. 77–97. Springer, Heidelberg (2009)
3. Berlekamp, E.R.: Goppa codes. IEEE Trans. Information Theory IT-19(3), 590–592 (1973)
4. Bernstein, D.J., Lange, T.: ebacs: Ecrypt benchmarking of cryptographic systems (February 17, 2009), <http://bench.cr.yp.to>
5. Bernstein, D.J., Lange, T., Peters, C.: Attacking and Defending the McEliece Cryptosystem. In: Buchmann, J., Ding, J. (eds.) PQCrypto 2008. LNCS, vol. 5299, pp. 31–46. Springer, Heidelberg (2008)
6. Biswas, B., Herbert, V.: Efficient root finding of polynomials over fields of characteristic 2. In: WEWoRC 2009, July 7-9 (2009)
7. Biswas, B., Sendrier, N.: McEliece crypto-system: A reference implementation, <http://www-rocq.inria.fr/secret/CBCrypto/index.php?pg=hymes>
8. Bogdanov, A., Eisenbarth, T., Rupp, A., Wolf, C.: Time-Area Optimized Public-Key Engines: $\mathcal{MQ}$ -Cryptosystems as Replacement for Elliptic Curves? In: Oswald, E., Rohatgi, P. (eds.) CHES 2008. LNCS, vol. 5154, pp. 45–61. Springer, Heidelberg (2008)
9. Chang, K.: I.B.M. Researchers Inch Toward Quantum Computer. New York Times Article (February 28, 2012), http://www.nytimes.com/2012/02/28/technology/ibm-inch-closer-on-quantum-computer.html?_r=1&hpw
10. Chien, R.: Cyclic decoding procedure for the bose-chaudhuri-hocquenghem codes. IEEE Trans. Information Theory IT-10(10), 357–363 (1964)

11. Cover, T.: Enumerative source encoding 19(1), 73–77 (1973)
12. Dinh, H., Moore, C., Russell, A.: McEliece and Niederreiter Cryptosystems That Resist Quantum Fourier Sampling Attacks. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 761–779. Springer, Heidelberg (2011)
13. ECRYPT. Yearly report on algorithms and key sizes (2007–2008). Technical report, D.SPA.28 Rev. 1.1 (July 2008),
<http://www.ecrypt.eu.org/documents/D.SPA.10-1.1.pdf>
14. Eisenbarth, T., Güneysu, T., Heyse, S., Paar, C.: MicroEliece: McEliece for Embedded Devices. In: Clavier, C., Gaj, K. (eds.) CHES 2009. LNCS, vol. 5747, pp. 49–64. Springer, Heidelberg (2009)
15. Faugere, J.-C., Otmani, A., Perret, L., Tillich, J.-P.: Algebraic cryptanalysis of mceliece variants with compact keys (2009)
16. Fischer, J.-B., Stern, J.: An Efficient Pseudo-random Generator Provably as Secure as Syndrome Decoding. In: Maurer, U.M. (ed.) EUROCRYPT 1996. LNCS, vol. 1070, pp. 245–255. Springer, Heidelberg (1996)
17. Güneysu, T., Paar, C., Pelzl, J.: Special-purpose hardware for solving the elliptic curve discrete logarithm problem. ACM Transactions on Reconfigurable Technology and Systems (TRETS) 1(2), 1–21 (2008)
18. Helion Technology Inc. Modular Exponentiation Core Family for Xilinx FPGA. Data Sheet (October 2008),
http://www.heliontech.com/downloads/modexp_xilinx_datasheet.pdf
19. Heyse, S.: Low-Reiter: Niederreiter Encryption Scheme for Embedded Microcontrollers. In: Sendrier, N. (ed.) PQCrypto 2010. LNCS, vol. 6061, pp. 165–181. Springer, Heidelberg (2010)
20. Heyse, S.: Implementation of McEliece Based on Quasi-dyadic Goppa Codes for Embedded Devices. In: Yang, B.-Y. (ed.) PQCrypto 2011. LNCS, vol. 7071, pp. 143–162. Springer, Heidelberg (2011)
21. Heyse, S., Moradi, A., Paar, C.: Practical Power Analysis Attacks on Software Implementations of McEliece. In: Sendrier, N. (ed.) PQCrypto 2010. LNCS, vol. 6061, pp. 108–125. Springer, Heidelberg (2010)
22. Horner, W.G.: A new method of solving numerical equations of all orders, by continuous approximation. Philosophical Transactions of the Royal Society of London 109, 308–335 (1819)
23. Huber, K.: Note on decoding binary Goppa codes. Electronics Letters 32, 102–103 (1996)
24. Kobara, K., Imai, H.: Semantically Secure McEliece Public-Key Cryptosystems-Conversions for McEliece PKC. In: Kim, K.-C. (ed.) PKC 2001. LNCS, vol. 1992, pp. 19–35. Springer, Heidelberg (2001)
25. McEliece, R.J.: A Public-Key Cryptosystem Based On Algebraic Coding Theory. Deep Space Network Progress Report 44, 114–116 (1978)
26. Misoczki, R., Barreto, P.S.L.M.: Compact McEliece Keys from Goppa Codes. In: Jacobson Jr., M.J., Rijmen, V., Safavi-Naini, R. (eds.) SAC 2009. LNCS, vol. 5867, pp. 376–392. Springer, Heidelberg (2009)
27. Molter, H., Stöttinger, M., Shoufan, A., Strenzke, F.: A simple power analysis attack on a McEliece cryptoprocessor. Journal of Cryptographic Engineering 1, 29–36 (2011), doi:10.1007/s13389-011-0001-3
28. Moradi, A., Kasper, M., Paar, C.: Black-Box Side-Channel Attacks Highlight the Importance of Countermeasures - An Analysis of the Xilinx Virtex-4 and Virtex-5 Bitstream Encryption Mechanism. In: Dunkelman, O. (ed.) CT-RSA 2012. LNCS, vol. 7178, pp. 1–18. Springer, Heidelberg (2012)
29. Niederreiter, H.: Knapsack-Type Cryptosystems and Algebraic Coding Theory. Problems of Control and Information Theory 15, 159–166 (1986)

30. Nojima, R., Imai, H., Kobara, K., Morozov, K.: Semantic security for the McEliece cryptosystem without random oracles. *Des. Codes Cryptography* 49(1-3), 289–305 (2008)
31. Patterson, N.: The algebraic decoding of Goppa codes. *IEEE Transactions on Information Theory* 21, 203–207 (1975)
32. Persichetti, E.: Compact McEliece keys based on quasi-dyadic srivastava codes. *Cryptology ePrint Archive*, Report 2011/179 (2011), <http://eprint.iacr.org/>
33. Cayrel, P.-L.: Code-based cryptosystems: implementations, <http://www.cayrel.net/research/code-based-cryptography/code-based-cryptosystems/>
34. Sendrier, N.: Efficient Generation of Binary Words of Given Weight. In: Boyd, C. (ed.) *Cryptography and Coding 1995*. LNCS, vol. 1025, pp. 184–187. Springer, Heidelberg (1995)
35. Sendrier, N.: Encoding information into constant weight words. In: *Proc. International Symposium on Information Theory, ISIT 2005*, September 4–9, pp. 435–438 (2005)
36. Shor, P.W.: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comput.* 26(5), 1484–1509 (1997)
37. Shoufan, A., Strenzke, F., Gregor Molter, H., Stöttinger, M.: A Timing Attack against Patterson Algorithm in the McEliece PKC. In: Lee, D., Hong, S. (eds.) *ICISC 2009*. LNCS, vol. 5984, pp. 161–175. Springer, Heidelberg (2010)
38. Shoufan, A., Wink, T., Molter, H.G., Huss, S.A., Strenzke, F.: A Novel Processor Architecture for McEliece Cryptosystem and FPGA Platforms. In: *20th IEEE International Conference on Application-specific Systems, Architectures and Processors* (July 2009)
39. Strenzke, F.: A Timing Attack against the Secret Permutation in the McEliece PKC. In: Sendrier, N. (ed.) *PQCrypto 2010*. LNCS, vol. 6061, pp. 95–107. Springer, Heidelberg (2010)
40. Strenzke, F., Tews, E., Gregor Molter, H., Overbeck, R., Shoufan, A.: Side Channels in the McEliece PKC. In: Buchmann, J., Ding, J. (eds.) *PQCrypto 2008*. LNCS, vol. 5299, pp. 216–229. Springer, Heidelberg (2008)
41. Sugiyama, Y., Kasahara, M., Hirasawa, S., Namekawa, T.: An erasures-and-errors decoding algorithm for goppa codes (corresp.). *IEEE Transactions on Information Theory* 22, 238–241 (1976)
42. van Tilborg, H.C.: *Fundamentals of Cryptology*. Kluwer Academic Publishers (2000)
43. Xilinx Inc. Advanced Security Schemes for Spartan-3A/3AN/3A DSP FPGAs, http://www.xilinx.com/support/documentation/white_papers/wp267.pdf
44. Xilinx Inc. Data Sheets and Product Information for Xilinx Spartan and Virtex FPGAs, <http://www.xilinx.com/support/>