A New Approximation Method for Constant Weight Coding and Its Hardware Implementation

Jingwei Hu and Ray C.C. Cheung

Additional information is available at the end of the chapter

http://dx.doi.org/10.5772/intechopen.75041

Abstract

In this chapter, a more memory-efficient method for encoding binary information into words of prescribed length and weight is presented. The solutions in existing work include complex float point arithmetic or extra memory overhead which make it demanding for resource-constrained computing platform. The solution we propose here solves the problems above yet achieves better coding efficiency. We also correct a crucial error in previous implementations of code-based cryptography by exploiting and tweaking the proposed encoder. For the time being, the design presented in this work is the most compact one for any code-based encryption schemes. We show, for instance, that our lightweight implementation of Niederreiter encrypting unit can encrypt approximately 1 million plaintexts per second on a Xilinx Virtex-6 FPGA, requiring 183 slices and 18 memory blocks.

Keywords: code-based cryptography, McEliece/Niederreiter cryptosystem, constant weight coding, FPGA implementation

1. Introduction

Most modern public-key cryptographic systems rely on either the integer factorization or discrete logarithm problem, both of which expect to be solvable on large-scale quantum computers using Shor's algorithm [1]. The recent breakthroughs of powerful quantum computing have shown their strength in computing solutions to the hard mathematical problems mentioned [2, 3]. The cryptographic research community has identified the urgency of insecure vulnerabilities rooted in these cryptosystems and begun to settle their security on alternative hard problems in the last years, such as multivariate-quadratic, lattice-based, and code-based cryptosystems [4]. In this chapter, we address the problem of encoding information into binary words of predefined length and Hamming weight in resource-constrained computing environment, e.g., reconfigurable hardware, embedded microcontroller systems, etc. This is of interest, in particular, of efficient implementations of McEliece's scheme [5, 6] or Niederreiter's scheme [7], the most prospective candidates for code-based cryptography.

In the case of Niederreiter/McEliece encryption, the binary stream of plaintext is requested to be converted into the form of constant weight words. Constant weight means that there exists a constant number of "1" in the binary plaintext. Note that in the hybrid Niederreiter encryption systems (KEM/DEM encryption) [8–10], KEMs are designed to exchange symmetric keys securely, and DEMs use these symmetric keys for transmitting long messages. This class of encryption techniques does not get constant weight coding involved. Nevertheless, if we want to construct a complete code-based cryptography including standard public-key encryption, digital signature, and hybrid encryption, constant weight coding must be efficiently implemented as it is required by both public-key encryption [11, 12] and signature [13]. The exact solution [14] of constant weight coding needs to compute large binomial coefficients and has a quadratic complexity though it is assumed to be optimal as a source coding algorithm. In [15], Sendrier proposed the first solution of linear complexity by incorporating Huffman codes. Later, the author of [15] further improves its coding efficiency very close to 1 by means of Goloumb's run-length encoding [16]. The new proposal is particularly easy to implement on hardware and has linear time complexity. The disadvantage though is that the length encoding is variable [17]. He also proposed an approximation version in this paper by regulating the value of d to the power of two. This approach significantly simplifies the encoding procedure and thus improves coding throughput.

Heyse et al. [18, 19] continued the research and proposed to adapt the approximation method in [17] to embedded system applications. Their design is implemented on AVR microcontrollers and Xilinx FPGAs. However, we observe that such method is not applicable to large parameter sets for the Niederreiter scheme (see **Table 1**) [20–22]. The work in [18, 19] preserves a lookup table of pre-stored data with the space complexity of $\mathcal{O}(n)$ to encode input messages into constant weight words. The memory overhead of this table is still intolerable for small embedded systems, and therefore their design is unscalable if n is large. CFS signature scheme [20] exploits very large Goppa code, and it requires to compress the lengthy constant weight signatures into a binary stream. MDPC-McEliece/Niederreiter encryption [23] uses very large n for practical security levels. For instance, n is set as large as 19,712 for 128-bit security and 65,536 for 256-bit security. Baldi et al. proposed a novel LDGM sparse syndrome signature scheme [13] with compact key size, which also requests a large constant weight coding within the signature generation. His method was successfully attacked in 2016 by a French research group [24]. At the time being, we do not have a considerably lightweight yet efficient solution for the constant weight encoding if we consider realizing such encoding in real-world applications.

The purpose of this work is to tweak Sendrier's approximation method [17] and hence to make it easy to implement for all possible secure parameters of the Niederreiter cryptosystem proposed in literature while maintaining the efficiency. Our contributions include:

1. We propose a new approximation method of constant weight coding free from complicated float point arithmetic and heavy memory footprint. This method permits us to implement a compact yet fast constant weight encoder on the resource-constrained computing platform.

(n, t)	Security level	Application	Coding system	Public/secret key size (kbits)	Prestored data for CW coding (kbits)
$(1024, 38)$	60 bit	Encryption	Goppa code	239/151.4	4
$(2048, 27)$	80 bit	Encryption	Goppa code	507/108.3	8
$(2690, 57)$	128 bit	Encryption	Goppa code	913/182.4	10.5
$(6624, 117)$	256 bit	Encryption	Goppa code	7488/2268.5	25.9
$(65536, 9)$	80 bit	Signature	Goppa code	9000/1019	256
$(262144, 9)$	80 bit	Signature	Goppa code	40,500/4525	1024
$(1048576, 8)$	80 bit	Signature	Goppa code	160,000/20,025	4096
$(9800, 18)$	80 bit	Signature	LDGM code	936/—	19.1
$(24960, 23)$	120 bit	Signature	LDGM code	4560/—	58.4
$(46000, 29)$	160 bit	Signature	LDGM code	13,480/—	117.2
$(19712, 134)$	128 bit	Encryption	MDPC code	9.6/—	77
$(65536, 264)$	256 bit	Encryption	MDPC code	32/—	256

Table 1. Parameters recommended used in the Niederreiter cryptosystem, referenced partly from [13, 19, 23, 25].

2. We improve the coding efficiency by fine-tuning the optimal precision for computing the value of d, in comparison with other approximation methods. The experiments have shown that the performance of our new method is better than Heyse's approximate version [18] and even comparable to Sendrier's original proposal [17].

3. We integrate our design with the Niederreiter encryption and obtain a more compact result. We fix a critical security flaw of Heyse et al.'s Niederreiter encryptor [19]. Our secure implementation of Niederreiter encryptor can encrypt approximately 1 million plaintexts per second on a Xilinx Virtex-6 FPGA, requiring 183 slices and 18 memory blocks.

This chapter is organized as follows. Sendrier's proposal of constant weight coding and its approximation variant [17, 18] is first revisited in Section 2. After analyzing the downside of these schemes, we are motivated to propose a new approximation method and to fine-tune it for an optimal source coding performance, presented in Section 3. Our detailed implementations for the proposed constant weight encoder/decoder and Niederreiter encryption unit on FPGAs are described in Section 4 and Section 5. We present our experimental results compared with the state of arts in Section 6. Finally, Section 7 summarizes this chapter.

2. Sendrier's methods for constant weight coding

Sendrier presented an algorithm for encoding binary information into words of prescribed length and weight [17]. His encoding algorithm returns a t-tuple $(\delta_1, \delta_2, \ldots, \delta_t)$ in which δ_i s are the lengths of the longest strings of consecutive "0"s. This method is easy to implement and has linear complexity with a small loss of information efficiency. In this work, we unfold the recursive encoding and decoding algorithms originated from [17] and rewrite them in Algorithm 1 and Algorithm 2 [26].

Algorithm 1: Encode Binary String to Constant Weight Word (Bin2CW) [26]

```
Input: message length $n$, message weight $t$ and a binary stream $B$
Output: a $t$-tuple $\Delta = (\delta_1, \delta_2, \ldots, \delta_t)$
$\delta = 0, index = 1$
while $t > 0$ do
    if $n \leq t$ then
        $t{-}{-}, n{-}{-}$
        $\delta_{index} = \delta$
        $\delta = 0, index{+}{+}$
    else
        $d =$ best_d$(n, t)$
        if $read(B, 1) = 1$ then
            $n {-}{=} d$, $\delta {+}{=} d$
        else
            $i =$ decodefd$(d, B)$
            $\delta_{index} = \delta + i$
            $n {-}{=} (i + 1), t{-}{-}, \delta = 0, index{+}{+}$
return $(\delta_1, \ldots, \delta_t)$
```

Algorithm 2: Decode Constant Weight Word to Binary String (CW2Bin) [26]

```
Input: $n$, $t$ and a $t$-tuple $\Delta = (\delta_1, \delta_2, \ldots, \delta_t)$
Output: a binary string $B$
$index = 1$
while $t \mathrel{!}= 0$ and $n > t$ do
    $d =$ best_d$(n, t)$
    if $\delta_{index} \geq d$ then
        $n {-}{=} d$, $\delta_{index} {-}{=} d$
        write$(B, 1)$
    else
        write$(B, 0)$
        write$(B,$ encodefd$(\delta_{index}, d))$
        $n {-}{=} (\delta_{index} + 1), t{-}{-}, index{+}{+}$
return $B$
```

We use the same notations from [17, 26] in the above two algorithms to keep consistency. For example, read(B, i) moves forward and reads i bits in the stream B and returns the integer whose binary decomposition has been read, most significant bit first; Write(B, i) moves forward and writes the binary string i into the stream B; and encodefd(δ_{index}, d) returns a binary string and decodefd(d, B) returns an integer. These two functions are actually the run-length encoding and decoding methods proposed by Golomb [16, 17]. best_d(n, t) returns an integer such that $1 \leq$best_d$(n, t) \leq n - t$ and Sendrier suggested to choose it close to the number defined by Eq. (1). In fact, best_d(n, t) can take any value in the range though the efficiency would be reduced if this value is too far from Eq. (1):

$$d = \left(n - \frac{t-1}{2}\right)\left(1 - \frac{1}{2^{\frac{1}{t}}}\right) \tag{1}$$

Sendrier also presented an approximation of the best d where the values of d (given by Eq. (1)) was restricted to the power of two [17]. More precisely, d is first computed via Eq. (1) and then round to $2^{\lceil \log_2(d) \rceil}$. This approximation greatly simplifies the functions of encodefd$(\cdot)$ and decodefd$(\cdot)$ and therefore outperforms in speed, while the loss of coding efficiency is trivial. The simplified versions of encoding and decoding with encodefd$(\cdot)$ and decodefd$(\cdot)$ after approximation are described as follows [26]:

$$\text{encodefd}(\delta, d) = base_2(\delta, u) \tag{2}$$

$$\text{decodefd}(d, B) = read(B, u) \tag{3}$$

where $base_2(\delta, u)$ denotes the u least significant bits of the integer δ written in base 2 and $u = \lceil \log_2(d) \rceil$. For the above two equations, the minimum allowed value of d is noteworthy in the case of $d = 1$. In this case we have $u = 0$, and therefore we define by purpose that $base_2(\delta, 0) = null$ and $read(B, 0) = 0$ to guarantee that our algorithm applies to all possible u.

Recently, Heyse et al. implemented Niederreiter encryption scheme on embedded microcontrollers in which they used a lookup table to compute the value of d for constant weight encoding [18]. Their method is based on the approximation method from [17]. One major contribution of their work is they observe that the last few bits of n can be ignored for constant weight encoding because these bits make little difference to the value of d. They do not keep $n \cdot t$ entries but instead n entries; the least significant $\lceil \log_2 t \rceil$ bits of n are not considered and are substituted by t. This method significantly reduces the size of the lookup table. According to our analysis, the lookup table is shrunk to roughly $\mathcal{O}(n)$. It works pretty well for small parameters of n, for example, $n = 2^{11}$ in the applications of Goppa code-based McEliece or Niederreiter encryption schemes. However, we occasionally found that it does not work well when we were implementing a Niederreiter signature scheme, called CFS signature. CFS requires an extremely large value of n, typically $n = 2^{18}, n = 2^{20}$. On the one hand, the size of lookup table increases linearly with n, resulting in somewhat unscalability. On the other hand, the coding efficiency drops dramatically and thus lowers the throughput of the constant weight encoder as n increases. All these downsides motivate us to figure out better ways of computing d. We would describe and analyze our new methods in the next section.

3. Proposed approximation method of *d*

3.1. Reduce memory footprint and computational steps

The computation of the value of d is the most crucial step of constant weight encoding and decoding, as suggested by Eq. (1) which involves floating-point arithmetic. However, many embedded/hardware systems do not have dedicated floating-point units for such computations.

[19] proposed to replace floating-point units by a lookup table with predefined data for reconfigurable hardware. The problem of their method is that, for some large (n, t), the lookup table could be sizeable. For example, $\left(n = 2^{16}, t = 9\right)$ requests the size of lookup table to be 256 kb, which is obviously not a negligible memory overhead for embedded systems.

To solve this problem, we propose to eliminate such lookup table by computing d directly using fixed-point arithmetic. We separate the computation of d into two parts. In the first part, $\theta[t] = \left(1 - 1/2^{\frac{1}{t}}\right)$ is precomputed and stored in the fixed-point format. In the second part, $\left(n - \frac{t-1}{2}\right) \cdot \theta$ is then efficiently calculated by a fixed-point multiplier. In this fashion we notably shrink the size of lookup table from $\mathcal{O}(n)$ to $\mathcal{O}(t)$ [26].

Furthermore, we substitute $\left(n - \frac{t-1}{2}\right)$ by n due to the following observations [26]:

- $n \gg t$ such that $n - \frac{t-1}{2} \approx n$.
- Eventually d must be round to an integer, and hence the difference between $\left(n - \frac{t-1}{2}\right)\theta$ and $n\theta$ is very likely to be ignored.

This substitution enables the removal of the computational steps of $n - (t-1)/2$, and hence a faster and simpler realization of constant coding which makes use of a single integer multiplication is achievable.

In summary, our new proposal of the approximation of d is as follows [26]:

$$d = \lfloor n \cdot \theta[t] \rfloor \tag{4}$$

where $\theta[t] = \left(1 - 1/2^{\frac{1}{t}}\right)$ is a function of t and precomputed. Our new approximation of d is lightweight, requiring only one multiplication. In the following, we will demonstrate that this method also permits reasonable high-coding efficiency as a source coding algorithm.

3.2. Represent $\theta[t]$ in fixed-point format

As aforementioned, $\theta[t]$ is actually a vector of fractional numbers and should be stored in fixed-point format. Note that the integer part of $\theta[t]$ is always 0, and therefore we only need to preserve its fractional part. Hereafter, we denote our fixed-point format as fixed_0_i, where i indicates the bits we have used for storing the fractional part of $\theta[t]$.

In practice, we hope to use fixed_0_i with the smallest i while maintaining desirable coding efficiency [26]. Smaller i means lower data storage and a simpler multiplication with smaller operand size, which is particularly advisable for resource-constrained computing platforms. Indeed, one of the key issues of this chapter is to determine the best fixed_0_i for $\theta[t]$. In the next section, we describe our experiments on exploring the optimal fixed_0_i.

3.3. Find the optimal precision for constant weight encoding

The purpose of the precision tuning is to find the lowest precision that still maintains a relatively high coding efficiency. A lower precision means we can use a multiplier of smaller operand size leading to better path delay and slice utilization. A higher coding efficiency means one can encode more bits from the source into a constant weight word. This is of interest, in particular, when someone encrypts a relatively large file using code-based crypto: It takes much less time for encryption if we have a high coding efficiency close to 1 [26].

To find the optimal precision of fixed_0_i, we studied the relationship between distinct *i*'s and their coding performance. In our experiments, all possible precision from fp_0_32 to fp_0_1 are investigated to compare with the Sendrier's methods [17] and Heyse's approximate version [18].

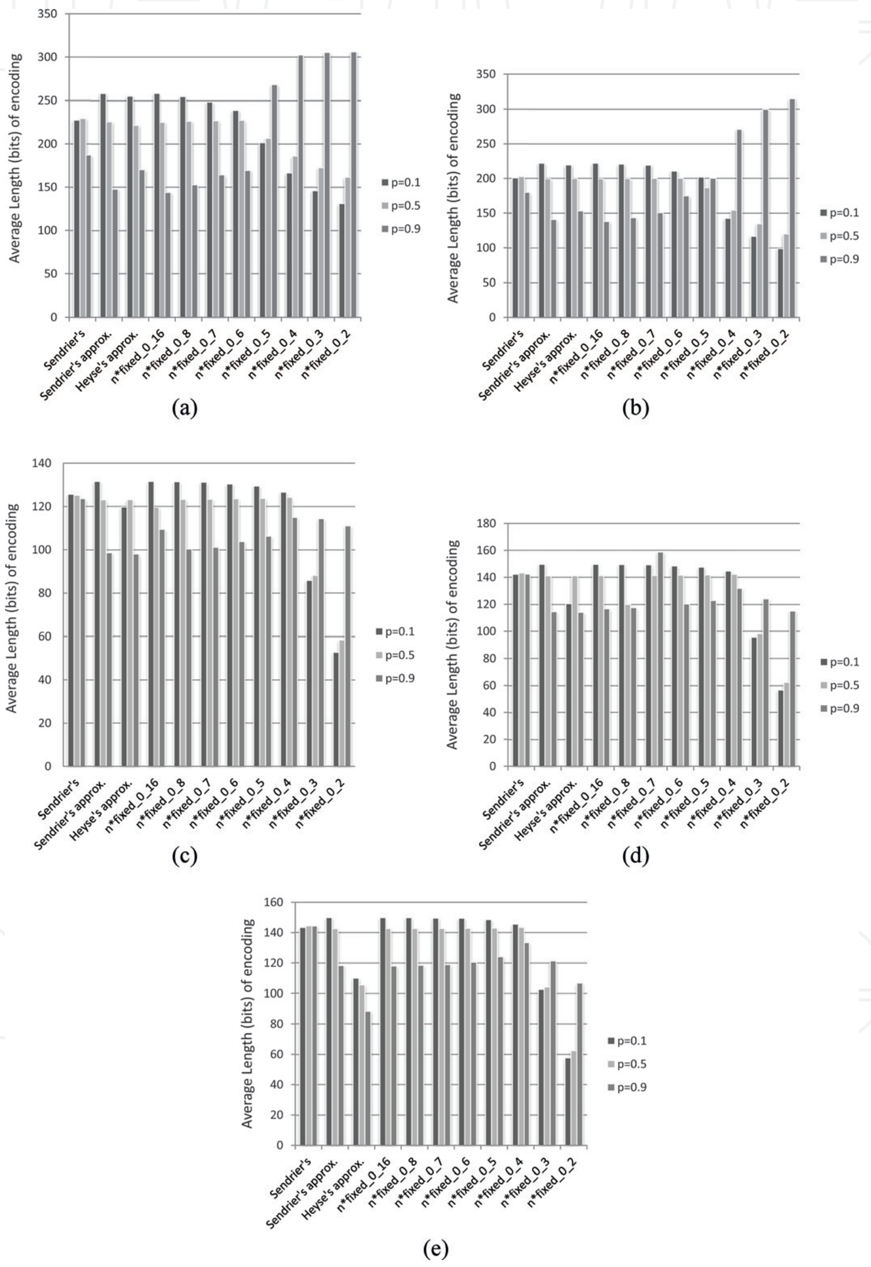

Figure 1. The performance of different methods for choosing the optimal d. We have listed five most frequently used sets of (n, t) for the Niederreiter cryptosystem. We have done three experiments for each (n, t) in which the input binary message contains "1" with the probability $p = 0.1$, 0.5, and 0.9, respectively. The results of each experiment are obtained by running 10,000 different messages. The X-axis lists different methods including Senderier's primitive [17], Senderier's approximation [17], Heyse's approximation [19], and our n*fixed_0_16 — N*fixed_0_2. The Y-axis represents the average length (bits) of the input message read for a successful constant weight encoding.

We measured the coding efficiency by calculating the average coding length for a successful encoding because longer coding length indicates a better coding efficiency. Since constant weight coding is variable length coding, we must consider how different plaintexts as input could affect the performance in order to determine which approximation is the best. To be thorough, different types of binary plaintexts, classified by the proportion of "1" contained, should be tested for evaluating the real performance of different encoding approximation methods. In our instances, we measure three particular types to simplify the model: "0" dominated texts (p = 0.1, "1" exists with probability of 0.1 in the plaintext), balanced texts (p = 0.5, "1" exists with probability of 0.5), and "1" dominated texts (p = 0.9, "1" exists with probability of 0.9) (**Figure 1**).

Figure 2 describes the coding performances when we adjust the precision of $\theta[t]$. Taken as a whole, the p = 0.1 group and the p = 0.5 group have a similar trend of average message length encoded as the arithmetic precision decreases: The message length drops slightly from n*fixed_0_16 — n*fixed_0_2 in consistency. On the contrary, the p = 0.9 group appears to be quite different where the numbers of bits read for a single constant weight coding first stay stable and

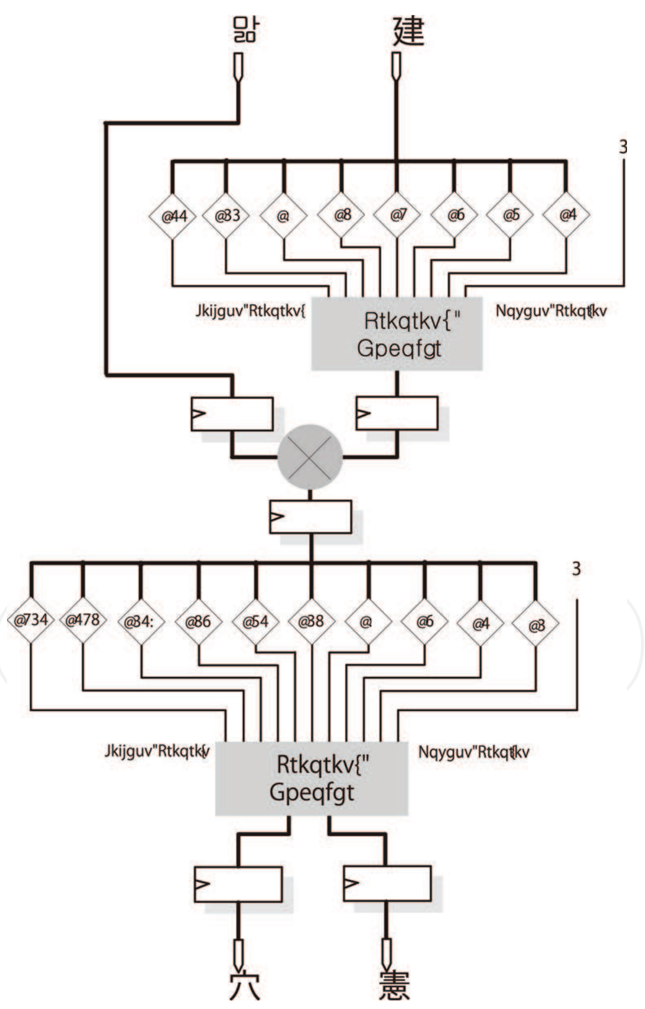

Figure 2. best_d module for CW encoder and decoder. We list here the detailed configurations of $n = 2^{11}, t = 27$ for demonstrative purpose.

then drop with the approximation precision decreasing. The numbers first keep stable because the loss of precision in $\theta[t]$ is comparatively trivial but if the precision drops too low, for instance, with fixed_0_2 representation $\theta[t] = 0$ for $2 \leq t \leq 38$ and hence $d = 0$. It leads to a constant n and small value of i in Algorithm 3 forcing us to read more bits of input stream before the algorithm halts. According to the evaluation criteria mentioned in the last paragraph, we compute the average length of the three types of plaintexts and identify the best approximation of d from our proposal after analyzing the statistics obtained. On the one hand, the n*fixed_0_5 group outperforms at $(n = 2^{10}, t = 38)$ and $(n = 2^{11}, t = 27)$. On the other hand, the n*fixed_0_4 group beats the others at $(n = 2^{16}, t = 9)$, $(n = 2^{18}, t = 9)$, and $(n = 2^{20}, t = 8)$.

Table 2 compares our proposed methods with the Sendrier's [17], Sendrier's original approximation using power of 2 [17], and Heyse's table lookup approximation [18]. From this table, it is seen that our proposal gains better coding efficiency than the original approximation and Heyse's approximation among all five parameter sets used for the Niederreiter scheme. Note that the average number of bits we have to read before producing the constant weight words is

n	t	Method	Number of bits read			Coding efficiency	Efficiency improved
			Maximum	Minimum	Average		
2^{10}	38	Sendrier's [17]	236	164	214.70	93.19%	—
		Sendrier's approx. [17]	263	124	210.45	91.32%	−1.98%
		Heyse's approx. [19]	261	143	210.73	91.44%	−1.85%
		n*fixed_0_5	311	183	225.54	97.89%	**+5.05%**
2^{11}	27	Sendrier's [17]	208	160	194.46	95.51%	—
		Sendrier's approx. [17]	227	119	187.55	92.11%	−3.56%
		Heyse's approx. [19]	224	127	190.96	93.78%	−1.80%
		n*fixed_0_5	361	132	196.30	96.41%	**+0.95%**
2^{16}	9	Sendrier's [17]	133	113	124.87	99.50%	—
		Sendrier's approx. [17]	133	77	117.80	93.84%	−5.10%
		Heyse's approx. [19]	133	86	116.34	92.68%	−6.83%
		n*fixed_0_4	135	95	121.98	97.20%	**−2.37%**
2^{18}	9	Sendrier's [17]	148	132	142.61	99.38%	—
		Sendrier's approx. [17]	151	91	135.17	94.18%	−5.22%
		Heyse's approx. [19]	300	101	133.21	92.81%	−6.60%
		n*fixed_0_4	154	112	139.64	97.31%	**−2.08%**
2^{20}	8	Sendrier's [17]	149	135	144.00	99.52%	—
		Sendrier's approx. [17]	151	99	136.94	94.64%	−4.90%
		Heyse's approx. [19]	265	17	110.07	76.07%	−23.56%
		n*fixed_0_4	158	109	140.81	97.31%	**−2.22%**

Table 2. The coding performance of the optimal d chosen from our approximation method.

upper bound by $\log_2\binom{n}{t}$, and the thus the ratio of the average number read and the upper bound measures the coding efficiency [17]. Additionally, our proposal even outperforms the Sendrier's method at two of these parameter sets—$(n = 2^{10}, t = 38)$ and $(n = 2^{11}, t = 27)$ with 5.05 and 0.95% of improvements, respectively. It is also worth mentioning that for $(n = 2^{16}, t = 9)$, $(n = 2^{18}, t = 9)$, and $(n = 2^{20}, t = 8)$, the performance of our proposal falls slightly behind with 2.37, 2.08, and 2.22% of loss when compared with the Sendrier's method; it nonetheless outruns Sendrier's approximation and Heyse's approximation. In particular, the performance of Heyse's approximation becomes unfavorable with 23.56% loss at $(n = 2^{20}, t = 8)$, and we are pushing the limits of Heyse's method here as the lower bits of n are innegligible and cannot be removed with such large n.

4. Proposed constant weight encoder and decoder

4.1. best_d module

The best_d module is the most critical arithmetic unit which computes the best value of d according to the inputs n and t. Our proposal of computation of *best_d* consists of three stages which performs the following task in sequence:

1. **Compute $\theta[t]$ via a priority encoder.** As discussed in Section 3, format fixed_0_5 is chosen to represent $\theta[t]$ for $(n = 2^{10}, t = 38)$ and $(n = 2^{11}, t = 27)$, and fixed_0_4 is used for

Value of t	$\theta[t] = (0.\theta_1\theta_2\theta_3\theta_4\theta_5)_2$*	$\theta[t] = (0.\theta_1\theta_2\theta_3\theta_4)_2$†
$22 \le t \le 38$	00000	N/A
$11 \le t \le 21$	00001	
$8 \le t \le 10$	00010	0001
$6 \le t \le 7$	00011	
$t = 5$	00100	0010
$t = 4$	00101	
$t = 3$	00110	0011
$t = 2$	01001	0100
$t = 1$	10,000	1000

*This $\theta[t]$ is represented in fixed_0_5 form, e.g., $\theta[t] = \sum_{i=1}^{5} \theta_i \cdot 2^{-i}$. This format is used in $(n = 2^{10}, t = 38)$ and $(n = 2^{11}, t = 27)$.

†This $\theta[t]$ is represented in fixed_0_4 form, e.g., $\theta[t] = \sum_{i=1}^{4} \theta_i \cdot 2^{-i}$. This format is used in $(n = 2^{16}, t = 9)$, $(n = 2^{18}, t = 9)$, and $(n = 2^{20}, t = 8)$.

Table 3. Encoding of $\theta[t]$.

$(n = 2^{16}, t = 9)$, $n = 2^{18}$, $t = 9$, and $n = 2^{20}$, $t = 8$. We notice via analysis that for some t, the values of $\theta[t]$ are identical. For instance, $\theta[6] = \theta[7] = (0.00011)_2 = 0.09375$ in fixed_0_5 format, shown in **Table 3**. This observation inspires us to exploit priority encoder to streamline the encoding of $\theta[t]$.

2. **Compute $n \cdot \theta[t]$ via a fixed-point multiplier.** Xilinx LogiCORE IP is configured to implement high-performance, optimized multipliers for different pairs of n and t. The fractional part of the multiplication result is truncated, but its integer part is preserved for the next stage to process.

3. **Output the value of d and u.** Recall that the value of $n \cdot \theta[t]$ must be round to $d = 2^u$. Another priority encoder is utilized to decode the integer part of $n \cdot \theta[t]$. The detailed decoding process is structured as lookup table mapping illustrated in **Table 4**.

Integer part of $n \cdot \theta[t]$	Value of d	Value of u
$n\theta[t] > 2^{18}$	2^{19}	19
$2^{17} < n\theta[t] \le 2^{18}$	2^{18}	18
$2^{16} < n\theta[t] \le 2^{17}$	2^{17}	17
$2^{15} < n\theta[t] \le 2^{16}$	2^{16}	16
$2^{14} < n\theta[t] \le 2^{15}$	2^{15}	15
$2^{13} < n\theta[t] \le 2^{14}$	2^{14}	14
$2^{12} < n\theta[t] \le 2^{13}$	2^{13}	13
$2^{11} < n\theta[t] \le 2^{12}$	2^{12}	12
$2^{10} < n\theta[t] \le 2^{11}$	2^{11}	11
$2^{9} < n\theta[t] \le 2^{10}$	2^{10}	10
$2^{8} < n\theta[t] \le 2^{9}$	2^{9}	9
$2^{7} < n\theta[t] \le 2^{8}$	2^{8}	8
$2^{6} < n\theta[t] \le 2^{7}$	2^{7}	7
$2^{5} < n\theta[t] \le 2^{6}$	2^{6}	6
$2^{4} < n\theta[t] \le 2^{5}$	2^{5}	5
$2^{3} < n\theta[t] \le 2^{4}$	2^{4}	4
$2^{2} < n\theta[t] \le 2^{3}$	2^{3}	3
$2 < n\theta[t] \le 2^{2}$	2^{2}	2
$1 < n\theta[t] \le 2$	2	1
$n\theta[t] \le 1$	1	0

Table 4. Decoding of $n \cdot \theta[t]$.

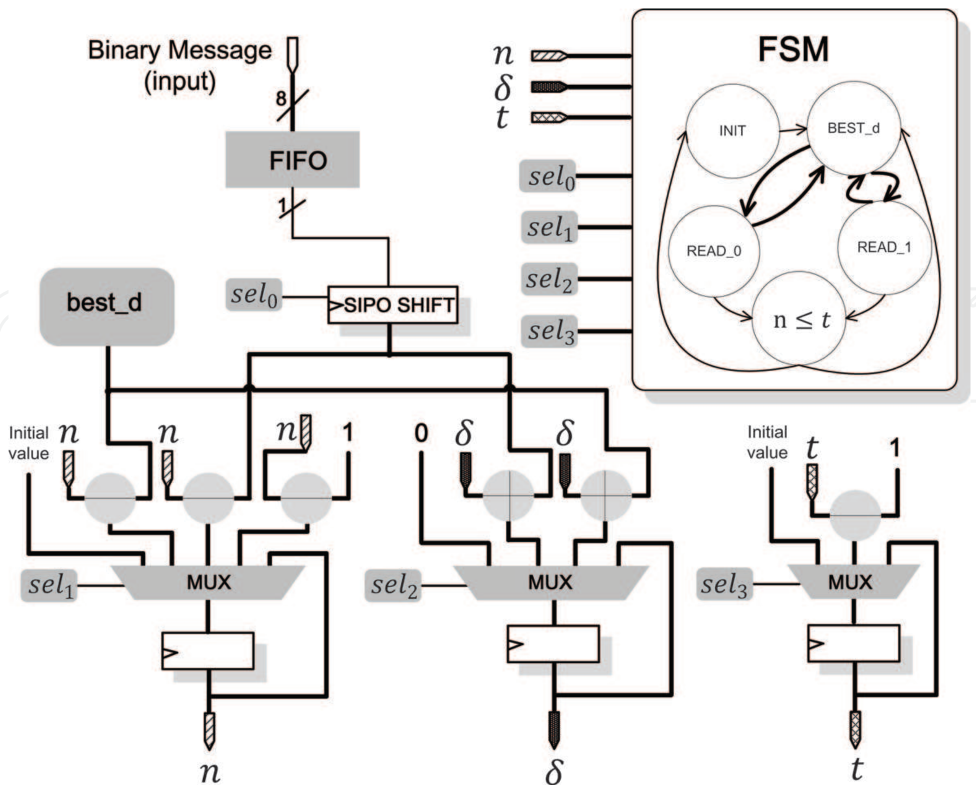

Figure 3. General architecture of CW encoder.

Figure 2 depicts our best_d unit. This unit works in three-stage pipelines. It first computes $\theta[t]$ and then obtains $n \cdot \theta[t]$ using a multiplier. Finally, the value of d would be determined by a priority decoder.

4.2. Bin2CW encoder

Figure 3 overviews the architecture of the proposed constant weight encoder. Input binary message is passed inward the encoder by means of nonsymmetric 8-to-1 FIFO-read which exactly imitates the function of $\text{read}(B, 1)$. A serial-in-parallel-out shift register is instantiated to perform $\text{read}(B, u), 0 \leq u \leq \left\lceil \log_2\left(\frac{n}{2}\right) \right\rceil$. The proposed best_d module is exploited here to calculate the value of d. The values of n, t, and δ are accordingly refreshed using three separate registers.

4.3. CW2Bin decoder

Figure 4 renders the architecture of the proposed constant weight decoder. A symmetric m-to-m bit FIFO is used to read the input t-tuple word by word. This logic is indeed the bottleneck of the constant weight decoder when compared with the encoder. Three registers are utilized to update the values of n and t as the Bin2CW encoder does, δ. The major difference is that the shift register here outputs the value of δ bit by bit as step 9 of Algorithm 2 demands.

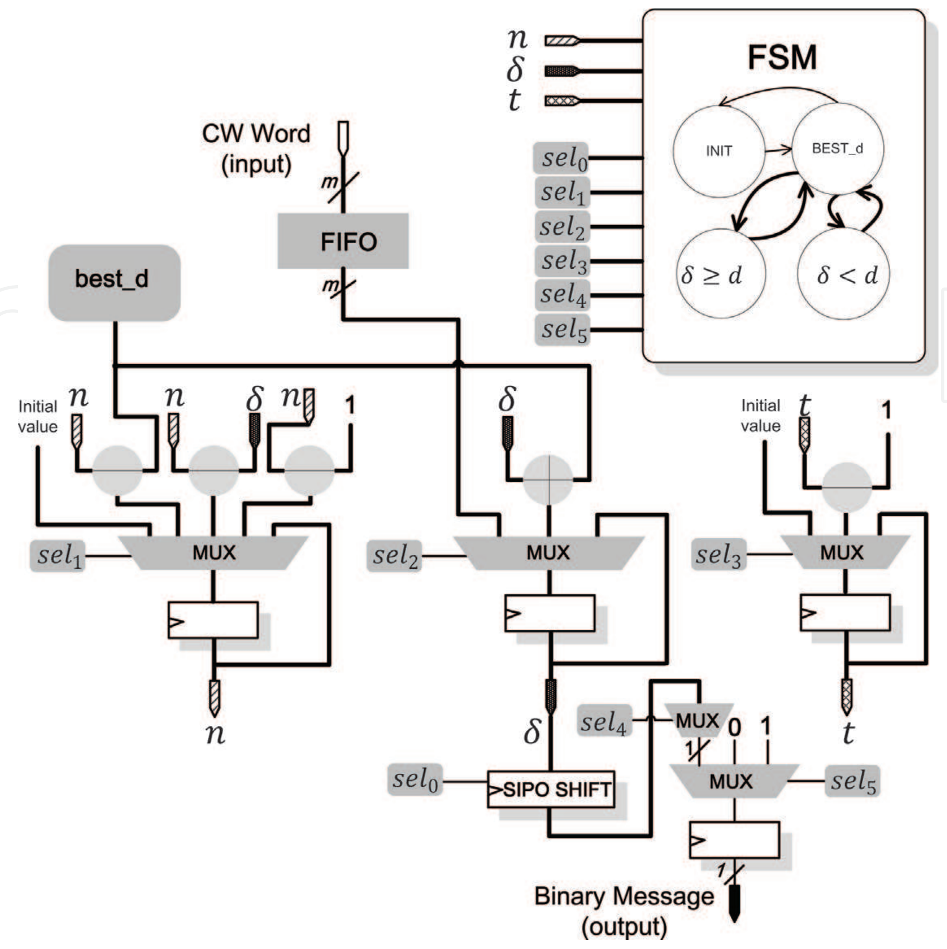

Figure 4. General architecture of CW decoder.

5. Integrating with the Niederreiter encryptor

In this section, we demonstrate that the proposed Bin2CW encoder can integrate into the Niederreiter encryptor for data encryption, shown in Algorithm 3.

Algorithm 3. Niederreiter Message Encryption, referenced from [25]

```
    Input: message vector $m$, public key $pk = \{\widehat{H}, t\}$ where $\widehat{H}$ is an $n$ by $mt$ matrix
    Output: ciphertext $c$
1   Bob encodes the message $m$ as a binary matrix/vector of length $n$ and weight at most $t$.
2   Bob computes the ciphertext as $c = \widehat{H} m^T$; $m^T$ is the transpose of matrix $m$.
3   return $c$.
```

The Bin2CW encoder performs the first step in Algorithm 3. Note that Bin2CW encoder returns a t-tuple of integers $(\delta_1, \ldots, \delta_t)$, which represents the distance between consecutive "1"s in the string. Nevertheless such t-tuple cannot be directly transported to compute the ciphertext. We believe that the way Heyse et al. [19] encrypts $c = \widehat{H}m^T$ with $m = (\delta_1, \ldots, \delta_t)$ is incorrect due to two reasons [26]:

1. It is very likely that $\delta_i = \delta_j$ where $i \neq j$ such that the number of errors is less than t, and it is assumed to be insecure for cryptanalysis.
2. $(\delta_1, \ldots, \delta_t)$ returns the integer ranging from 0 to $n - t$, but the constant weight word exactly ranges from 0 to n. In other terms, the last t rows of the public key $\widehat{H}^T$ are never used.

To correct this weakness from [19], we propose to generate the "real" constant weight binary words of length n and Hamming weight t. Assume the constant weight is represented by $(i_1, \ldots, i_t)$, the coordinates of the "1"s in ascending order, then $i_1 = \delta_1$, $i_2 = \delta_2 + i_1 + 1$, …, and $i_t = \delta_t + i_{t-1} + 1$ are computed as the input of the second step, Algorithm 3.

Figure 5 illustrates our revision of Niederreiter encryption unit on the basis of [19]. The public key $\widehat{H}^T$ is stored in an internal BRAM and row-wise addressed by the 11-bit register. Two 11-bit integer adders are instantiated to transform $(\delta_1, \ldots, \delta_t)$ to $(i_1, \ldots, i_t)$ which are eventually stored in the 11-bit register. The vector–matrix multiplication in step 2, Algorithm 3, is equivalent to XOR operation of the selected rows of $\widehat{H}^T$, which can be implemented as a $GF(2^{297})$ adder in this figure. It is also worth noting that the vector–matrix multiplication works concurrently with the CW encoding: Whenever a valid i_k has been computed, it is transferred immediately to the $GF(2^{297})$ adder for summing up the selected row. Once the last i_t has been

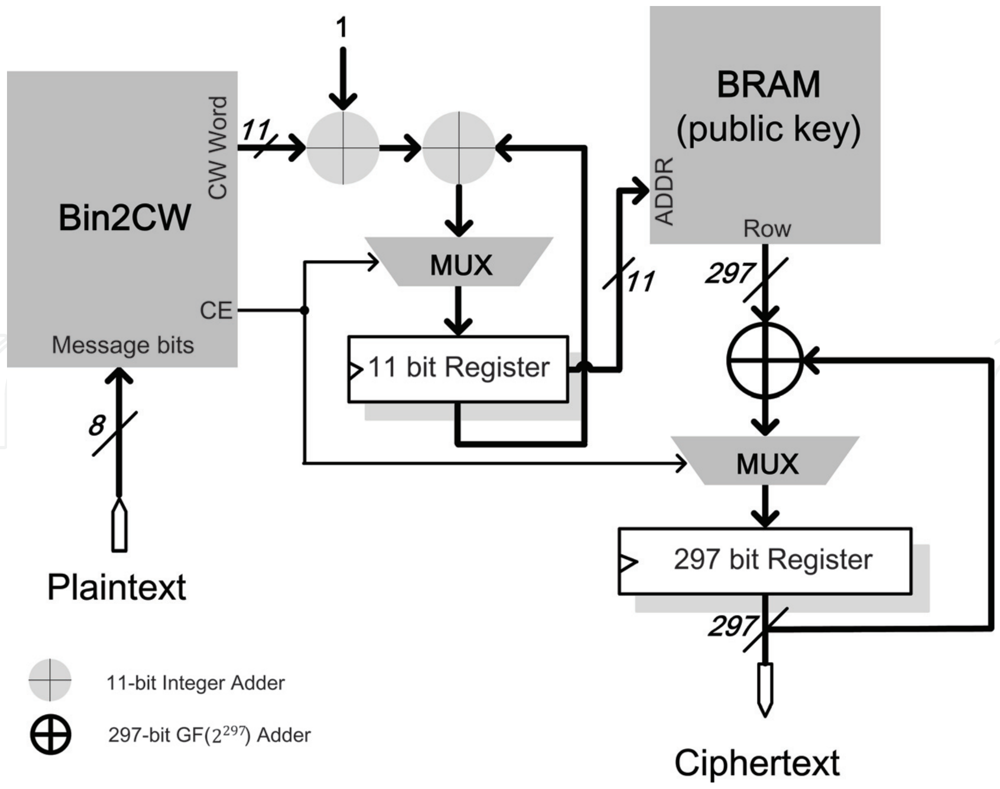

Figure 5. Block diagram of the Niederreiter encryptor.

computed, the last indexed row of $\widehat{H}^T$ also has been accumulated to the sum. This sum, stored in the 297-bit register, is now available to be interpreted as the ciphertext.

Our final remark is for the side channel attacks of code-based crypto using constant weight encoding (CWE). Admittedly, we cannot give a satisfying answer of them at the time being. We believe this is an open problem left to be solved. For the time being, if the users decide not to take the risk of timing attacks, we suggest forcing the CWE to be constant time. We can set the maximum time (it happens when we have all-zero input) for whatever the input is. Nevertheless, the price is a significant drop of timing performance.

We give here our analysis of timing attacks: The attackers can compromise the CWE if and only if he could analyze the timing differences among different inputs and use this information to recover the entire message. Unfortunately, the timing character of the operation of reading "1" or "0" is different: when reading "1," it consumes only 1 clock cycle count, whereas when reading "1," it continues to read $\lceil \log_2(d) \rceil$ more bits from B, consuming $1 + \lceil \log_2(d) \rceil$ counts. These behaviors appear at first sight to be vulnerable to timing attacks: For different inputs, the execution time is slightly different, and the distinction between reading "0" and reading "1" is also significant. However, statistical approaches as [27, 28] have introduced against RSA cryptosystems which seem to be not helpful: To our way of thinking, the situation we encounter is much more difficult. (1) We need to recover this particular message under attack, but the timing differences among different messages that we collect do not leak any useful information on the targeted message. On the contrary, in RSA we can use a large number of messages and compare the timing for recovering the secret key in a bit-by-bit fashion. (2) Note that d is changing each iteration according to the current state of n and t. That is to say, when reading "1," the timing is variable, sensitive to how "1" and "0" are permuted in the message and thus difficult to predict. Most importantly, even if CWE is somehow compromised, it does not reveal any information about secret keys. In the case of decryption, the ciphertext is first decrypted by an error correcting decoder (typically, Goppa-code or MDPC-code decoder) which holds the secret key. The result after error correcting is a $GF(2^n)$ vector, and then this vector is encoded by CWE for the plaintext recovered. We can see the key points here: Timing attacks should be mounted on error correcting decoders rather than constant weight decoders for retrieving the secret keys. Perhaps a better strategy is to mount timing attacks on CWE for recovering the plaintext directly. This raises one more question: how do we distinguish or measure the peculiar timing of CWE out of the total execution time, given that error correcting decoders also take nonconstant time for decoding? This is indeed a very exciting topic for which we would investigate in our future work.

6. Results and comparisons

We captured our constant weight coding architecture in the Verilog language and prototyped our design on Xilinx Virtex-6 FPGA (**Table 5**). The reason why we did our experiments on Xilinx Virtex-6 is principally about a convention. Recent progress in FPGA implementations of

	Algorithm	Platform	Area [slices]	Memory blocks [36 + 18 kb RAM]	Frequency [MHz]	Throughout [Mbps]	Coding efficiency
(a) $n = 2^{10}$, $t = 38$							
This work	Bin2CW, new approx.	Xilinx xc6vlx240t	74	0 + 1	330	160.2	97.89%
	CW2Bin, new approx.		79	0 + 1	330	148.1	
Heyse et al. [19]	Bin2CW, Heyse's approx.	Xilinx xc6vlx240t	110	0 + 1	310	178.8	91.44%
	CW2Bin, Heyse's approx.		88	0 + 1	310	162.1	
(b) $n = 2^{11}$, $t = 27$							
This work	Bin2CW, new approx.	Xilinx xc6vlx240t	91	0 + 1	350	187.2	96.41%
	CW2Bin, new approx.		95	0 + 1	340	168.6	
Heyse et al. [19]	Bin2CW, Heyse's approx.	Xilinx xc6vlx240t	118	0 + 1	340	208.4	93.78%
	CW2Bin, Heyse's approx.		110	0 + 1	340	164.6	
Sendrier [17][*]	Bin2CW, original	Intel Pentium 4	—	—	2400	17.3	95.51%
	Bin2CW, approximate					33.0	92.11%
(c) $n = 2^{16}$, $t = 9$							
This work	Bin2CW, new approx.	Xilinx xc6vlx240t	103	0 + 1	440	316.1	97.20%
	CW2Bin, new approx.		109	0 + 1	310	212.9	
Heyse et al. [19]	Bin2CW, Heyse's approx.	Xilinx xc6vlx240t	90	10 + 3	240	182.3	92.68%
	CW2Bin, Heyse's approx.		90	10 + 3	230	170.3	
Sendrier [17]	Bin2CW, original	Intel Pentium 4	—	—	2400	18.3	99.50%
	Bin2CW, approximate					22.0	93.84%
(d) $n = 2^{18}$, $t = 9$							
This work	Bin2CW, new approx.	Xilinx xc6vlx240t	138	0 + 1	410	295.5	97.31%
	CW2Bin, new approx.		118	0 + 1	320	219.5	
Heyse et al. [19]	Bin2CW, Heyse's approx.	Xilinx xc6vlx240t	94	40 + 3	170	106.0	92.81%
	CW2Bin, Heyse's approx.		93	40 + 3	180	104.9	

	Algorithm	Platform	Area [slices]	Memory blocks [36 + 18 kb RAM]	Frequency [MHz]	Throughout [Mbps]	Coding efficiency
(e) $n = 2^{20}$, $t = 8$							
This work	Bin2CW, new approx.	Xilinx xc6vlx240t	156	0 + 1	370	284.9	97.31%
	CW2Bin, new approx.		122	0 + 1	300	222.7	
Heyse et al. [19]	Bin2CW, Heyse's approx.	Xilinx xc6vlx240t	124	160 + 3	130	96.6	76.07%
	CW2Bin, Heyse's approx.		125	160 + 3	130	98.8	

*Sendrier implemented a different but very close parameter set $n = 2^{11}$, $t = 30$. We also put it here for reference.

Table 5. Compact implementations of CW encoder and decoder on Xilinx Virtex-6 FPGA.

code-based cryptography accept Virtex-6 or even lower ends for implementation aspects [19, 29–32]. The benefit of using Virtex-6 from our standpoint is that we could fairly compare our design with others given that most of them are also implemented on Virtex-6. To the best of our knowledge, the only compact implementations of constant weight coding have been proposed by Heyse et al. [19]. Their lightweight architecture is generally identical to ours except the design of best_d module. Their best_d module works in two pipeline stages: In the first stage, it retrieves the value of u by table lookup. Then in the second stage, it outputs d according to the value of u using a simple decoder. Comparatively, our best_d module has three stages of the pipeline, and thus it leads to a lower throughput, but our architectures are smaller and improve the area-time tradeoff of the constant weight coding implementations proposed by Heyse et al. [19], shown in **Table 5**. In particular, we use only one 18 kb memory block for all parameter sets of our experiments.

We also observe that in our designs, the memory footprint does not increase, and the high clock frequency also maintains as the parameters grow. This is because the main difference among encoders or decoders with distinct parameters n and t is the data width of multiplier embedded in the best_d module, which increases logarithmically from $10bit \times 5bit$ to $20bit \times 4bit$. On the other hand, the memory overhead of Heyse's implementations grows linearly with n and might introduce problems when n is large as aforementioned. To verify this argument, we re-implemented Heyse's work for $(n = 2^{16}, t = 9)$, $(n = 2^{18}, t = 9)$, and $(n = 2^{20}, t = 8)$. The experimental results validate this point. Additionally, another negative side effect of heavy memory overhead is that the working frequency of circuits drops rapidly as shown in **Table 6**. For small parameters (a) and (b), the lookup table in Heyse's design could be made of distributed memory (LUT) and therefore has little impact on frequency. However, for large parameters (c), (d), and (e), such lookup table can no longer be instantiated as LUTs because Xilinx Virtex-6 distributed memory generator only allows maximum data depth of 6,5536. We instead use block memory resource of the FPGAs to construct the table, and this accordingly hinders speed performance due to relatively far and complicated routing. The usage of block memory is the real bottleneck of Heyse's work as n grows.

Aspect (Virtex6-VLX240)	Niederreiter [12]	This work
Slices	315	183
LUTs	926	505
FFs	875	498
BRAMs	17	18[*]
Frequency	300 MHz	340 MHz
CW encode $e = \mathrm{Bin2CW}(m)$	≈200 cycles	349.1 cycles
Encrypt $c = e \cdot \widehat{H}$	≈200 cycles	352.1 cycles

[*]We used 16×36kb RAMs and 2×16kb RAMs.

Table 6. FPGA implementation results of Niederreiter encryption with $n = 2048, t = 27$ compared with [19] after PAR.

We finally implemented the Niederreiter encryptor, a cryptographic application where constant weight coding is used exactly as described in Section 5. **Table 6** compares our work with the state of art [19, 26]. It is seen that our new implementation is the most compact, with better area-time tradeoffs. The same amount of block memory is occupied in our design as [19] did where 16 × 36kb + 1 × 18kb RAMs are utilized to save the public-key matrix $\widehat{H}$ and one 18 kb RAM for the 8-to-1 FIFO within the constant weight encoder.

7. Conclusion

A new approach for determining the optimal value d in constant weight coding is proposed in this chapter. This method innovates a more compact yet efficient architecture for constant weight encoder and decoder in resource-constrained computing systems. Afterward, we exploited this new encoder to implement the Niederreiter encryptor on a Xilinx device. Experiments show that our work competes for the state of art and works better in terms of both RAM usage and processing throughput for large parameters.

Author details

Jingwei Hu* and Ray C.C. Cheung

*Address all correspondence to: jw.hu@cityu.edu.hk

Department of Electronic Engineering, City University of Hong Kong, Hong Kong

References

[1] Shor PW. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM Journal on Computing. 1997;**26**(5):1484-1509

[2] Vandersypen LM, Steffen M, Breyta G, Yannoni CS, Sherwood MH, Chuang IL. Experimental realization of Shor's quantum factoring algorithm using nuclear magnetic resonance. Nature. 2001;**414**(6866):883-887

[3] Xu N, Zhu J, Lu D, Zhou X, Peng X, Du J. Quantum factorization of 143 on a dipolar-coupling nmr system. arXiv preprint arXiv:1111.3726; 2011

[4] Bernstein DJ. Introduction to post-quantum cryptography. In: Post-Quantum Cryptography. Berlin: Springer; 2009. pp. 1-14

[5] McEliece RJ. A public-key cryptosystem based on algebraic coding theory. DSN Progress Report. 1978;**42**(44):114-116

[6] Sendrier N. Code-based public-key cryptography. In: Post-Quantum Cryptography Summer School. 2014

[7] Niederreiter H. Knapsack-type cryptosystems and algebraic coding theory. Problems Of Control and Information Theory-Problemy Upravleniya I Thorii Informatsii. 1986;**15**(2): 159-166

[8] Persichetti E. Secure and anonymous hybrid encryption from coding theory. In: International Workshop on Post-Quantum Cryptography. Springer; 2013. pp. 174-187

[9] von Maurich I, Heberle L, Güneysu T. Ind-cca secure hybrid encryption from qc-mdpc niederreiter. In: International Workshop on Post-Quantum Cryptography; Springer; 2016. pp. 1-17

[10] Chou T. Qcbits: Fast constant-time code-based cryptography. In: Cryptographic Hardware and Embedded Systems-CHES 2016; Springer. 2016. pp. 250-272

[11] Biswas B, Sendrier N. Mceliece cryptosystem implementation: Theory and practice. In: International Workshop on Post-Quantum Cryptography; Springer; 2008. pp. 47-62

[12] Overbeck R, Sendrier N. Code-based cryptography. In: Post-Quantum Cryptography. Springer; 2009. pp. 95-145

[13] Baldi M, Bianchi M, Chiaraluce F, Rosenthal J, Schipani D. Using ldgm codes and sparse syndromes to achieve digital signatures. Post-quantum cryptography. Springer. 2013:1-15

[14] Cover TM. Enumerative source encoding. Information Theory, IEEE Transactions on. 1973;**19**(1):73-77

[15] Sendrier N. Efficient generation of binary words of given weight. Cryptography and Coding. Springer. 1995:184-187

[16] Goloumb G. Run length encoding. Information Theory, IEEE Transactions on. 1966;**12**: 399-401

[17] Sendrier N. Encoding information into constant weight words. In: Information Theory, 2005. ISIT 2005. Proceedings. International Symposium on. IEEE. 2005. pp. 435-438

[18] Heyse S. Low-reiter: Niederreiter encryption scheme for embedded microcontrollers. In: Post-Quantum Cryptography; Springer; 2010. pp. 165-181

[19] Heyse S, Güneysu T. Towards one cycle per bit asymmetric encryption: Code-based cryptography on reconfigurable hardware. In: Cryptographic Hardware and Embedded Systems–CHES 2012; Springer; 2012. pp. 340-355

[20] Courtois NT, Finiasz M, Sendrier N. How to achieve a Mceliece-based digital signature scheme. In: Advances in Cryptology-ASIACRYPT 2001; Springer; 2001. pp. 157-174

[21] Landais G, Sendrier N. CFS software implementation. IACR Cryptology ePrint Archive. 2012;**2012**:132

[22] Finiasz M. Parallel-CFS. In: Selected Areas in Cryptography. Berlin: Springer; 2011. pp. 159-170

[23] Misoczki R, Tillich J-P, Sendrier N, Barreto PS. Mdpc-mceliece: New mceliece variants from moderate density parity check codes. In: 2013 IEEE International Symposium on Information Theory Proceedings (ISIT); IEEE. 2013. pp. 2069-2073

[24] Phesso A, Tillich J-P. An efficient attack on a code-based signature scheme. In: International Workshop on Post-Quantum Cryptography; Springer; 2016. pp. 86-103

[25] Hu J, Cheung RC. An application specific instruction set processor (ASIP) for the Niederreiter cryptosystem. Cryptology ePrint Archive. Report 2015/1172; 2015. http://eprint.iacr.org/2015/1172.pdf

[26] Hu J, Cheung RC, Güneysu T. Compact constant weight coding engines for the code-based cryptography. IEEE Transactions on Circuits and Systems II: Express Briefs. 2017; **64**(9):1092-1096

[27] Kocher PC. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. Annual International Cryptology Conference. Springer. 1996:104-113

[28] Dhem J-F, Koeune F, Leroux P-A, Mestré P, Quisquater J-J, Willems J-L. A practical implementation of the timing attack. In: International Conference on Smart Card Research and Advanced Applications; Springer; 1998. pp. 167-182

[29] Heyse S, Von Maurich I, Güneysu T. Smaller keys for code-based cryptography: QC-MDPC Mceliece implementations on embedded devices. In: Cryptographic Hardware and Embedded Systems–CHES 2013; Springer; 2013. pp. 273-292

[30] Beuchat J-L, Sendrier N, Tisserand A, Villard G. FPGA Implementation of a Recently Published Signature Scheme. France Doctoral dissertation, INRIA; 2004

[31] Chen C, Eisenbarth T, Von Maurich I, Steinwandt R. Differential power analysis of a mceliece cryptosystem. In: International Conference on Applied Cryptography and Network Security. Springer; 2015. pp. 538-556

[32] von Maurich I, Güneysu T. Lightweight code-based cryptography: QC-MDPC McEliece encryption on reconfigurable devices. In: Proceedings of the Conference on Design, Automation & Test in Europe. European Design and Automation Association; 2014. p. 38