VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

Ke Liao

QR CODE GENERATION AND APPLICATION OF ANTI-FALSIFICATION

Technology and Communication

2016

PREFACE

It has been about two years since I started my student life in Vaasa University of Applied Sciences. I would like to express my greatest gratitude to VAMK for giving me such a great environment to study.
I would also like to thank my thesis supervisor Dr Chao Gao for providing me the opportunity to work on this exciting field of research. He has guided me throughout this project with great patience and offered me plenty of helpful suggestions along the way.
Having gone through quite a few obstacles, I realized the importance of patience working as an engineer. Even a tiny mistake with the code can make a great difference to the result.
Finally, I succeeded and fulfilled all my original expectation.

Ke Liao

Vaasa, Finland

04/05/2016

VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES
Information Technology

ABSTRACT

Author	Ke Liao
Title	QR Code Generation and Application of Anti-falsification
Year	2016
Language	English
Pages	34 + 10 Appendices
Name of Supervisor	Chao Gao

QR code is two-dimensional bar code. It is more advanced than bar code which can only store numbers and characters. QR code can store numbers, characters (including Chinese characters) and even images. It has great data capacity and allows for error correction. It is commonly used to spread information and is sometimes used to achieve website login function. It can even be encrypted so that the user has to enter a password to get the original information.
In this thesis, I come up with the idea that the QR code can be used for checking the genuineness of products. This is achieved by applying a CRC32 algorithm, which will produce a checksum value according to the original information so that check operation can be performed when we scanning a product's QR code and so tell whether this product is verified.
This is quite a practical implementation method because it is easy to operate.

[1]

[1] Key words QR Code, CRC32 Algorithm, Anti-falsification

CONTENTS

LIST OF FIGURES AND TABLES

LIST OF ABBREVIATIONS

CRC	Cyclic redundancy check
ECI	Extended Channel Interpretation
MD5	Message-digest algorithm
PDP	Position Detection Pattern
PHP	Professional Hypertext Pre-processor
QR Code	Quick Response Code
URI	Uniform Resource Identifier
URL	Uniform Resource Locator

LIST OF APPENDICES

1 INTRODUCTION

The introduction chapter introduces the motivation, the general structure of this project and the structure of the thesis report.

1.1 Motivation of the Project

Counterfeits threatens the interests of manufacturers and consumers, and they have a serious impact on the economy development of the country. Government and companies have to spend a lot of manpower and financial resources every year for the security crackdown. Many manufacturers lose a lot of interests when other manufacturers copy their products and sell fake products at a low price. Consumers also suffer from the spread of fake products the quality of which is always not guaranteed. The poor quality of fake products also ruins the brand reputation. Therefore, it is necessary to apply anti-falsification technology to help customers distinguish the difference between authentic products and counterfeits.

There are many different anti-falsification technologies. Consumers can check the genuineness of a product by uncovering the coating on the product and getting a random number which is pre-generated by algorithm. Then customers can dial a phone number to check if this random number is stored in the database. If so, then this product is verified, otherwise it is asserted as a counterfeit. Anti-falsification technology can also be achieved by inserting microchips to the wanted products. In this case, the product information is stored on a microchip and this chip can only be scanned by specific devices.

Now that the QR code has spread all over the world because of its fast readability and great storage capacity, it would be a good idea to apply to QR code to check the validity of products. It is easier to operate compared to dialling a number. Customers only need a smartphone, which is widespread today, to scan the QR codes printed on the products to get the validation information. It has a lower cost in comparison to inserting a microchip on the products. Actually, manufacturers only need trivial investment to apply the QR code anti-falsification technology.

However, the QR code is just a method to carry the information, it also need to think about how to perform the validation process. This is achieved by applying a CRC32 algorithm. By applying the CRC32 algorithm, we can get a checksum value corresponding to the original

information. A combined information of the original information and the checksum value is stored in the QR code and this QR code will be printed on the product. Hence, the check operation can be performed by scanning a product's QR code and so tell whether this product is verified.

1.2 Overview Structure of the Project

Figure 1 displays the structure of the thesis.

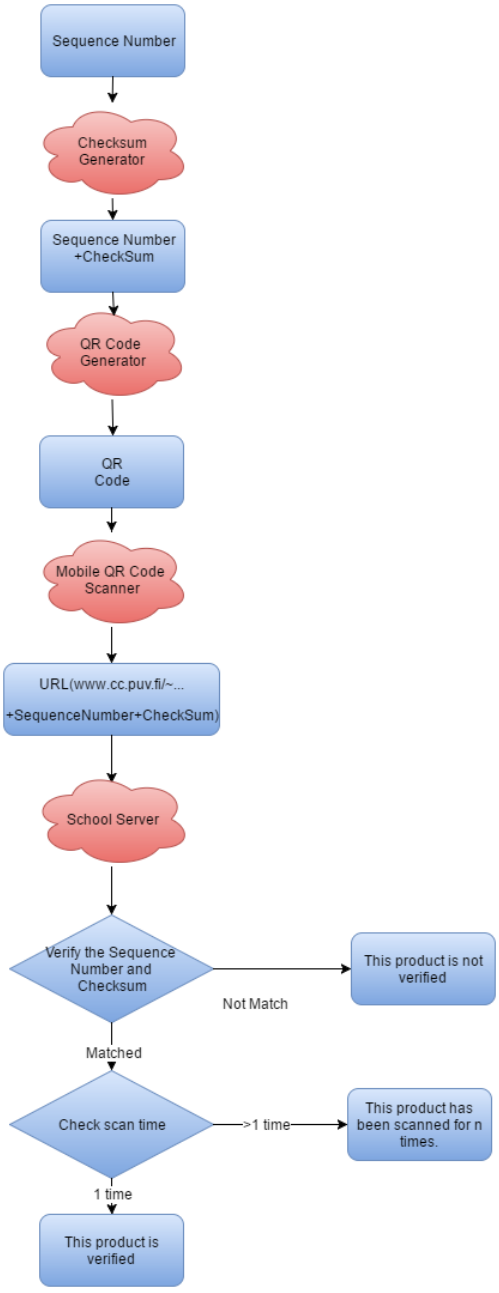

Figure 1.Project Structure

The product sequence number is sent to the checksum generator and a checksum value is generated based on the CRC32 algorithm, then the sequence number and the checksum value are sent to the QR code generator and are stored as part of QR code information. Next a QR code will be generated and it will be printed on the surface of product and the product is stored in a box.

A customer can download a QR code scanner to scan the QR code on the product. This QR code stores a URL. The URL is a HTTP query string which contains a product sequence number and its corresponding checksum. The query is sent to a verification server which runs the same algorithm as the QR code generator.

The server checks the sequence number and the checksum information and if they match, it will check whether this QR code has been scanned before, if not, a message which says this product is verified will show on the website. If it has been scanned before, the number of scan times will be returned.

Since the QR code generation is quite a complicated technology, I simply apply a JAVASCRIPT library QRCode.js to implement this function. It is published on GITHUB and is open-source. Therefore, it can be used without any prerequisite. On the generation side, there is another function which is the checksum generation function. It is based on the CRC32 algorithm and is written in JAVASCRIPT. On the server side, there is also a CRC32 checksum generation function written in PHP. We obtain the sequence number from the query string and generate a correspondent CRC32 remainder value and check if this value matches the checksum value (received from the query string). The reason why this checksum generator on the server side is written in PHP is so that hackers cannot figure out how our checksum value is generated.

The principle is easy to understand and implement. It also facilitates the process of checking the validity of products.

1.3 Overall Structure of the Thesis

The basic structure of the thesis is as follows:

Chapter 1 describes the motivation and overall architecture of the project. Chapter 2 introduces the related background knowledge about developing this project. Chapter 3 focuses on the implementation of the project in details, while the Chapter 4 illustrates the test procedure and result. Chapter 5 draws conclusions.

2 BACKGROUND

This chapter introduces the background knowledge of the technologies used in this project, including QR code, the QR code generation method and the CRC32 Algorithm.

2.1 QR Code

QR code is a two-dimensional matrix symbol and it consists of an array of square modules. It has three position detection patterns located at three corners of the symbol and they are intended to help locate its position, size and inclination. It also provides a wide range set symbol sizes and four levels of error correction/5/.

QR code has two different models, including model 1 and model 2. QR code model 1 is the original specification for the QR code. Model 2 is an enhanced form of the symbol with additional features and is more universal today. Model 2 is used in this thesis.

The QR code system was invented in 1994 by Denso Wave. It is widely used to track vehicles during manufacture and it is also applied in commercial tracking applications. Many mobile applications also include this technology because of its convenience. It can be used to display text or image to the user, to open a Uniform Resource Identifier (URI), or to send messages.

Individual users can generate QR codes by themselves to share information, manufacturers can apply QR code on the products to provide company information or publish advertisements. The technology has since become one of the most-used types of two-dimensional barcode/2/.

2.1.1 QR Code Symbol Structure

Each QR code symbol is constructed of nominally square modules and consists of an encoding region, function patterns, namely finder, separator, timings patterns and alignment patterns. It is surrounded by a quiet zone border on all sides. Figure 2 illustrates the structure of a Version 7 QR Code symbol.

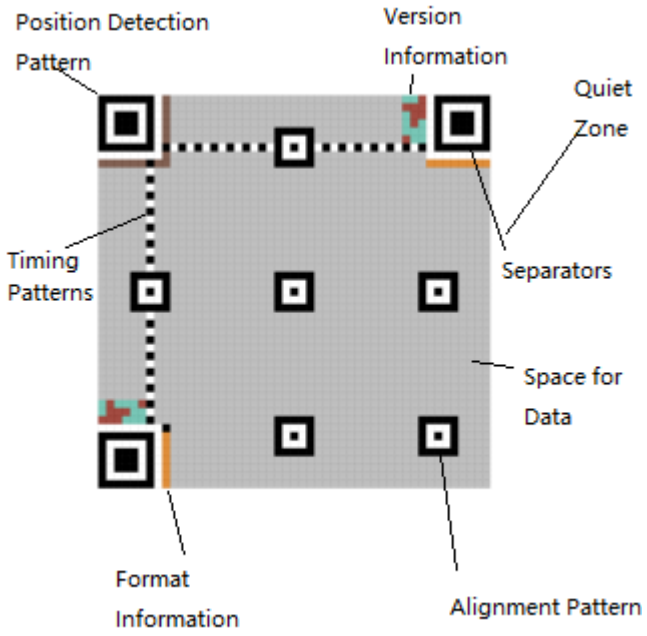

Figure 2.Structure of a QR Code symbol/5/

Representation of data

A module is the unit length of QR code (one square area comprising QR Code)/6/. A dark module is a binary one and a light module is a binary zero.

Quiet Zone

It is a 4X (X is the width of a QR code module) wide region which surrounds the symbol on four sides.

Finder Pattern

It consists of three Position Detection Patterns (PDP) located at the upper left, upper right and lower left corners of the QR code symbol. Each PDP consists of dark 7 × 7 modules, light 5 × 5 modules and dark 3 × 3 modules. Figure 3 shows the structure of the position detection pattern (finder pattern).

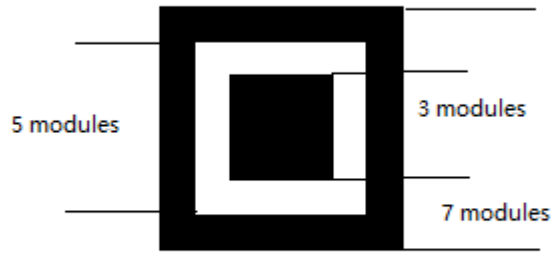

Figure 3.Structure of Position Detection Pattern/5/

Separators

As illustrated in Figure 2, a separator consists of all light modules and it is one-module wide. There is a separator between each finder pattern and encoding region.

Segment

Segment is a sequence of data encoded according to the rules of one ECI (Extended Channel Interpretation) or encoding mode.

Terminator

Terminator has a bit pattern of 0000 and it is used to end the data bit string.

Timing Patterns

The horizontal and vertical Timing Patterns respectively consist of a one module wide row or column of alternating dark and light modules, commencing and ending with a dark module.

Alignment Patterns

Alignment pattern consists of dark 5 × 5 modules, light 3 × 3 modules and a single central dark module. Different symbol version has different number of alignment patterns.

Format Information

Format information provides information of the symbol's error correction level and masking pattern.

Version Information

Version information is applied in QR code model 2. It contains information on the symbol version and the error correction bits for this data.

2.1.2 Symbol Versions and sizes

QR code has 40 versions and each version has different size. Version 2 is a matrix of 21 modules × 21 modules, version 2 measures 25 modules × 25 modules and so on increasing in steps of 4 modules per side up to Version 40 which measures 177 modules × 177 modules/5/. Figure 4 illustrates the structure of Versions 40 which has the highest data capacity. In this thesis, the version of the QR code is chosen based on the size of the data to carry.

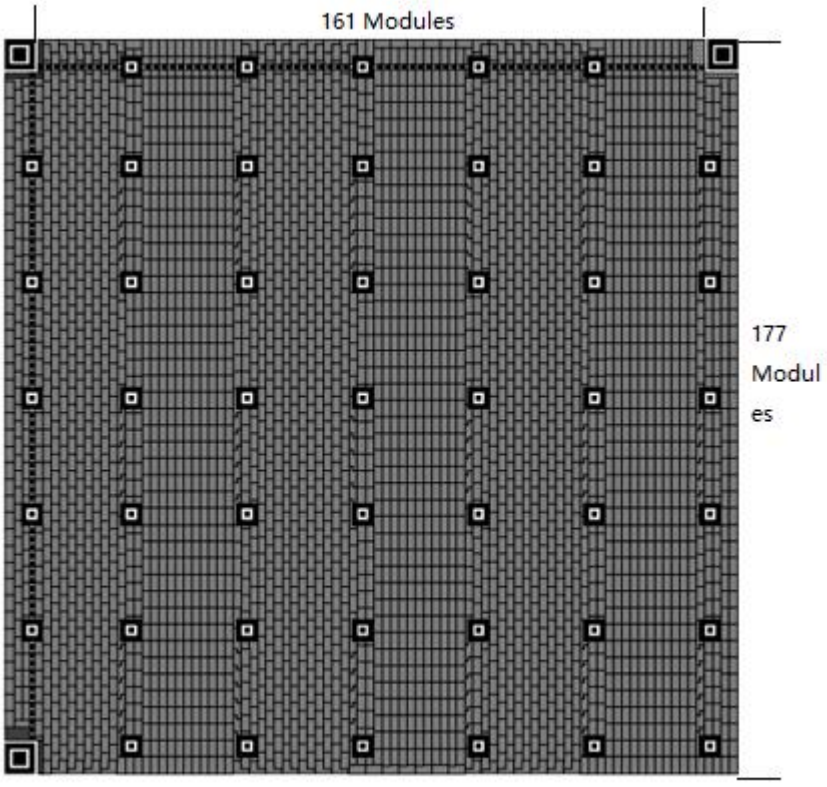

Figure 4.Version 40 Symbol/5/

Data Capacity

The amount of data that can be stored in the QR code symbol depends on the datatype (mode, or input character set), version (1, …, 40, indicating the overall dimensions of the symbol), and error correction level. The maximum storage capacities occur for 40-L symbols (version 40, error correction level L). Below Table 1 shows the data capacity for all versions of the QR code.

Version	No.of Modules/Side(A)	Function Pattern modules(B)	Format Version Information modules(C)	Data modules except (C)(D=$A^2 - B - C$)	Data capacity[$codewords$][3]	Remainder Bits
1	21	202	31	208	26	0
2	25	235	31	359	44	7
3	29	243	31	567	70	7
4	33	251	31	807	100	7
5	37	259	31	1079	134	7
6	41	267	31	1383	172	7
7	45	390	67	1568	196	0
8	49	398	67	1936	242	0
9	53	406	67	2336	292	0
10	57	414	67	2768	346	0
11	61	422	67	3232	404	0
12	65	430	67	3728	466	0
13	69	438	67	4256	532	0
14	73	611	67	4651	581	3
15	77	619	67	5243	655	3
16	81	627	67	5867	733	3
17	85	635	67	6523	815	3
18	89	643	67	7211	901	3
19	93	651	67	7931	991	3
20	97	659	67	8683	1085	3
21	101	882	67	9252	1156	4
22	105	890	67	10068	1258	4
23	109	898	67	10916	1364	4
24	113	906	67	11796	1474	4
25	117	914	67	12708	1588	4
26	121	922	67	13652	1706	4
27	125	930	67	14628	1828	4
28	129	1203	67	15371	1921	3
29	133	1211	67	16411	2051	3
30	137	1219	67	17483	2185	3
31	141	1227	67	18587	2323	3
32	145	1235	67	19723	2465	3
33	149	1243	67	20891	2611	3
34	153	1251	67	22091	2761	3
35	157	1574	67	23008	2876	0
36	161	1582	67	24272	3034	0
37	165	1590	67	25568	3196	0

38	169	1598	67	26896	3362	0
39	173	1606	67	28256	3532	0
40	177	1614	67	29648	3706	0

Table 1.Data Capacity of all versions of QR code /5/

Error Correction Capacity

The QR code allows for error correction and has four different error correction levels (L, M, Q, H).

QR code with a different error correction level has capacity of recovery from different amounts of damage.

As is shown in Table 2, the lowest error correction level is L and it allows for 7 percent of damage to be recovered. The highest error correction level is H and it allows for 30 percent of damage to be recovered. Error correction level H is selected for this thesis.

Table 2.Error Correction Levels /5/

Error Correction Level	Recovery Capacity % (approx..)
L	7
M	15
Q	25
H	30

Encoding Mode

There are many different encoding modes in QR code. Next is a list of the frequently used encoding modes.

1) Extended Channel Interpretation (ECI) Mode/5/

The input data is interpreted according to the default character set. Four broad types of interpretation are supported in QR Code:

a) international character sets (or code pages).

b) general purpose interpretations such as encryption or compaction.

c) user-defined interpretations for closed systems.

d) control information for structured append in unbuffered mode

2) Numeric Mode/5/

Numeric mode accepts decimal data and encodes the decimal digit set. Three data characters are represented by 10 bits. For example, 012 is represented as 0000001100.

3) Alphanumeric Mode/5/

Alphanumeric mode encodes data from a set of 45 characters, including ten decimal digits (0-9), 26 alphabetic characters (A-Z) and nine symbols (SP, $, %, *, +, -, ., /, :). Each character is represented by a corresponding ASCII value according to the ASCII table. Normally the two input characters are represented by 11 bits binary string. For example, with input AC, the corresponding index values (in Table 3) are 10 and 12 respectively. Then 10 is multiplied by 45 and plus 12, the result is 462, finally 462 is converted to 10 bits binary string 00111001110.

Table 3.Encoding/decoding table for Alphanumeric Mode/5/

Char.	Value	Char.	Value	Char.	Value	Char.	Value	Char.	Value	Char.	Value	Char.	Value	Char.	Value
0	0	6	6	C	12	I	18	O	24	U	30	SP	36	.	42
1	1	7	7	D	13	J	19	P	25	V	31	$	37	/	43
2	2	8	8	E	14	K	20	Q	26	W	32	%	38	:	44
3	3	9	9	F	15	L	21	R	27	X	33	*	39		
4	4	A	10	G	16	M	22	S	28	Y	34	+	40		
5	5	B	11	H	17	N	23	T	29	Z	35	-	41		

8-bit Byte Mode/5/

The 8-bit byte mode encodes the 8-bit Latin/Kana character set in accordance with JIS X 0201 (character values 00HEX to FFHEX). In this mode data is encoded at a density of 8 bits. Table 4 shows the mode indicators for all the modes.

Table 4.Mode Indicators/5/

Mode	Indicator
ECI	0111
Numeric	0001
Alphanumeric	0010
8-bit Byte	0100
Kanji	1000
Structured Append	0011

FNC1	0101 (First position) 1001 (Second position)
Terminator (End of Message)	0000

8-bit byte mode is used in this thesis because it can handle almost all kinds of characters as shown in Table 5.

Char.	Hex	Char.	Hex	Char.	Hex	Char.	Hex	Char.	Hex	Char.	Hex	Char.	Hex
NUL	00	SP	20	@	40	`	60		80		A0	タ	C0		E0
SOH	01	!	21	A	41	a	61		81	。	A1	チ	C1		E1
STX	02	"	22	B	42	b	62		82	「	A2	ツ	C2		E2
ETX	03	#	23	C	43	c	63		83	」	A3	テ	C3		E3
EOT	04	$	24	D	44	d	64		84	、	A4	ト	C4		E4
ENQ	05	%	25	E	45	e	65		85	・	A5	ナ	C5		E5
ACK	06	&	26	F	46	f	66		86	ヲ	A6	ニ	C6		E6
BEL	07	'	27	G	47	g	67		87	ァ	A7	ヌ	C7		E7
BS	08	(	28	H	48	h	68		88	ィ	A8	ネ	C8		E8
HT	09	)	29	I	49	I	69		89	ゥ	A9	ノ	C9		E9
LF	0A	*	2A	J	4A	j	6A		8A	ェ	AA	ハ	CA		EA
VT	0B	+	2B	K	4B	k	6B		8B	ォ	AB	ヒ	CB		EB
FF	0C	,	2C	L	4C	l	6C		8C	ャ	AC	フ	CC		EC
CR	0D	-	2D	M	4D	m	6D		8D	ュ	AD	ヘ	CD		ED
SO	0E	.	2E	N	4E	n	6E		8E	ョ	AE	ホ	CE		EE
SI	0F	/	2F	O	4F	o	6F		8F	ッ	AF	マ	CF		EF
DLE	10	0	30	P	50	p	70		90	ー	B0	ミ	D0		F0
DC1	11	1	31	Q	51	q	71		91	ア	B1	ム	D1		F1
DC2	12	2	32	R	52	r	72		92	イ	B2	メ	D2		F2
DC3	13	3	33	S	53	s	73		93	ウ	B3	モ	D3		F3
DC4	14	4	34	T	54	t	74		94	エ	B4	ヤ	D4		F4
NAK	15	5	35	U	55	u	75		95	オ	B5	ユ	D5		F5
SYN	16	6	36	V	56	v	76		96	カ	B6	ヨ	D6		F6
ETB	17	7	37	W	57	w	77		97	キ	B7	ラ	D7		F7
CAN	18	8	38	X	58	x	78		98	ク	B8	リ	D8		F8
EM	19	9	39	Y	59	y	79		99	ケ	B9	ル	D9		F9
SUB	1A	:	3A	Z	5A	z	7A		9A	コ	BA	レ	DA		FA
ESC	1B	;	3B	[	5B	{	7B		9B	サ	BB	ロ	DB		FB
FS	1C	<	3C	¥	5C	|	7C		9C	シ	BC	ワ	DC		FC
GS	1D	=	3D	]	5D	}	7D		9D	ス	BD	ン	DD		FD
RS	1E	>	3E	^	5E	‾	7E		9E	セ	BE	゛	DE		FE
US	1F	?	3F	_	5F	DEL	7F		9F	ソ	BF	゜	DF		FF

Table 5.Encoding/decoding table for JIS8 character set/5/

In this thesis, the generation of QR code is implemented by applying a JAVASCRIPT library which will be explained later.

2.2 Selection of the Algorithm

The selection of the algorithm is the most important part in the whole process. The initial plan was to protect the data stored in the QR code by a password so that the user cannot scan the

QR code successfully without knowing a password. However, this process is not user-friendly and it is hard to implement in the real process. There is also a cryptographic algorithm called md5, which will produce a more complicated result. But it cannot avoid the risk that the hacker figures out which algorithm is used. Finally, I came up with an idea that it can be achieved by applying a CRC32 algorithm, which can secure this process since a not commonly used polynomial value is applied and it will not be easy to crack this value.

2.3 CRC32 Algorithm

A Cyclic Redundancy Check (CRC) is a hash which is frequently used as a checksum for data in for instance archives. A checksum is the value calculated based on the original data by performing mod 2 polynomial division.

This method was invented by W. Wesley Peterson in 1961; the 32-bit CRC function of Ethernet and many other standards is the work of several researchers and the idea was published in 1975/3/.

There are different versions of CRC and a CRC is called an n-bit CRC when its check value is n-bits. In this project, CRC32 is used and the check value is 32 bits.

2.3.1 Polynomial

When applying a polynomial to CRC algorithms, the coefficients of a polynomial needs to be treated as the numbers in a bit-string.

For example, supposing there is a following 5-bits polynomial.

$$x^4 + x + 1$$

When treating the coefficients of this polynomial as the numbers in a bit-string, the following result is reached.

10011

CRC performs a module-2 division between the message data and the polynomial, if the polynomial is n bits long, the remainder of the binary division is n-1 bits long. A CRC32 has a 33-bit polynomial and the remainder is 32 bits long.

2.3.2 Choosing A Polynomial

The selection of the generator polynomial is the most important part of implementing the CRC algorithm. The polynomial must be chosen to maximize the error-detecting capabilities while minimizing overall collision probabilities/3/.

It is hard to choose an appropriate polynomial, however, there is a standard polynomial library which is universally used.

Table 6.CRC32 Standard Polynomials/3/

Name	**Normal Polynomial**	**Reversed Polynomial**
CRC-32	0x04C11DB7	0xEDB88320
CRC-32C	0x1EDC6F41	0x82F63B78
CRC-32K	0x741B8CD7	0xEB31D82E
CRC-32K2	0x32583499	0x992C1A4C
CRC-32Q	0x814141AB	0xD5828281

Table 6 displays the polynomials used for all versions of CRC32.

In this project, CRC-32 is used and as can be seen from Table 4, the standard polynomial of CRC-32 is 0x04C11DB7. However, in order to prevent some hackers from easily figuring out the polynomial value, there is small change to this standard polynomial. The last hex number is replaced with 6, which produces the final polynomial value 0x04C11DB6. Such a small change with the polynomial value will produce quite a different check value according to the CRC algorithm. For example, if the original data is 123, the CRC remainder would be 2286445522 with the polynomial value of 0x04C11DB7 and 3383897554 with the polynomial value of 0x04C11DB6.

2.3.3 A Straightforward CRC Implementation

The basic idea of CRC algorithms is simply to treat the message as an enormous binary number, to divide it by another fixed binary number (polynomial), and to make the remainder from this division the checksum. Upon receipt of the message, the receiver can perform the same division and compare the remainder with the "checksum"/4/, as shown in Figure 5.

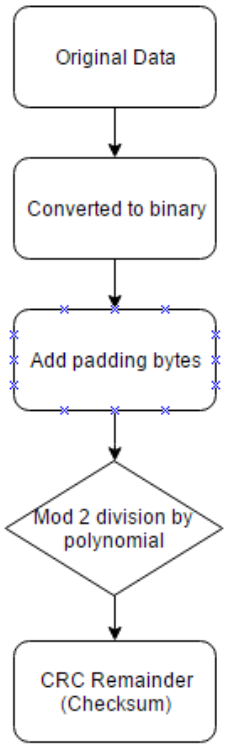

Figure 5.Checksum Generation /4/

Figure 5 explains how the checksum value is generated. First, there is original data, which is converted to a binary string. Then padding bits are added to the end of this binary string. The number of padding bits is based on the length of CRC. In this project, 32 bits of 0 are added to the binary string. Then mod 2 division is performed on this padded binary string so that we can get our CRC remainder value, which is the checksum.

2.3.4 "Reflected" Table-Driven Implementations

In the real process, a simple straightforward CRC implementation cannot suffice our need. The reason is that this simple implementation operates at the bit level, which is quite inefficient to

execute. Therefore, this "Reflected" Table-Driven Implementation method is introduced as it can speed up the process. This was done because most of the hardware chips at the time reversed data bitwise.

In the "Reflected" Table-Driven Implementation method, the operation level is the byte level. The following Figure 6 illustrates how this implementation works.

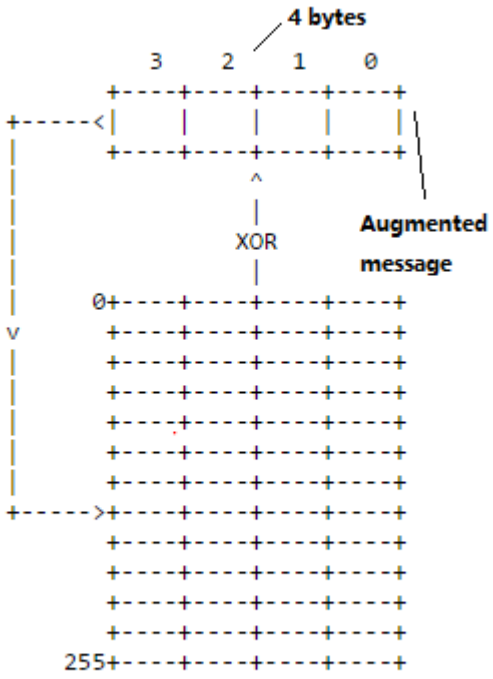

Figure 6. "Reflected" Table-Driven Implementation/4/

In the CRC32 table-driven implementation, the message is augmented into bytes string, and the register processes 4 bytes of data at a time. Each byte has 8 bits, which has a range between 0 and 255. Therefore, a table is pre-generated which stores all the CRC32 remainder values of numbers between 0 and 255. When performing mod 2 division on byte, it is possible to quickly refer to that table and get the remainder result. Finally, it is possible to get the CRC remainder value of the whole string.

2.3.5 "Reversed" Polynomials

In order to further improve the processing speed, so called “Reversed” Polynomials were introduced, the principle is quite simple. The polynomial is reversed means all of its bits are shifted from right to left. For example, supposing there is a polynomial with a corresponding binary string of 101011, then the reversed polynomial value will be 110101.

3 IMPLEMENTATION

This chapter introduces the implementation of the project, including the used QR Code Library, CRC32 Implementation and the Flowchart of the whole project.

3.1 QR Code Library

Since this QR Code generation is quite a complicated technology, it would be too difficult to achieve it by myself. Therefore, I applied a universal QR code generation library/1/ in this project and it is written in JAVASCRIPT. It is published under MIT license and the copyright is Copyright (c) 2012 davidshimjs. It is fully open-source and anyone can employ this library without any prerequisite. It consists of three JAVASCRIPT files, including jquery.min.js, jquery.qrcode.js and qrcode.js/1/.

In this library, the default error correction level is H. The version number of the generated QR code is dependent on how many bytes of data needs to be carried. The default encode mode is an 8-bit byte mode.

3.2 CRC32 Implementation

In the second chapter, how the CRC algorithm works was introduced together with several different implementation methods. However, it may still seem a little confusing so I will make a conclusion on how it works in the real process.

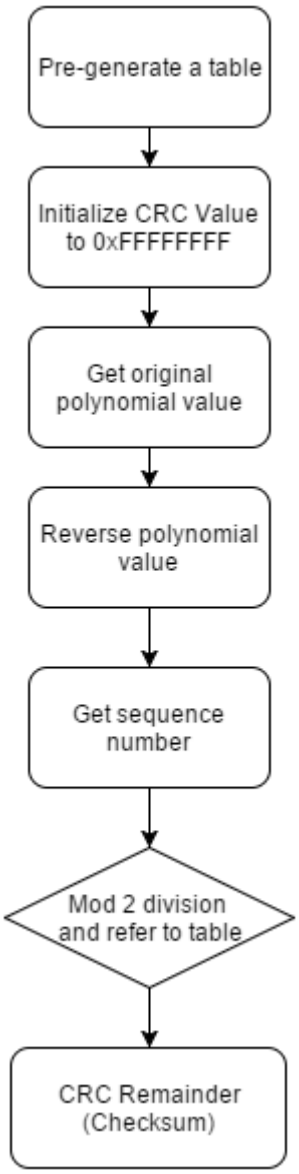

Figure 7. CRC32 Implementation

Figure 7 displays the whole process of the CRC32 implementation. According to a "Reflected" Table-Driven implementation which was explained in chapter 2, needs to be pre-generated. A table which stores all the CRC32 remainder(checksum) values of the numbers ranging between 0 and 255. Then the CRC initial value is set as a default value 0Xffffffff. After that, the polynomial value that we want to use is resolved, it is 0X04C11DB6 and after reversing this polynomial we get 0X6DB88320. The next step is to perform the mod 2 division on the original sequence number value by the reversed polynomial value, this is done by dividing the sequence number to bytes and all the bytes are processed in a loop and the table needs to be referred to in this step to optimize the processing speed. Finally, the CRC remainder value is reached.

3.3 Flowchart of the Program

In this project, there is a CRC32 sender and a CRC32 receiver. The CRC sender generates a QR code and the CRC receiver verifies the information stored in the QR code.

3.3.1 CRC32 Sender

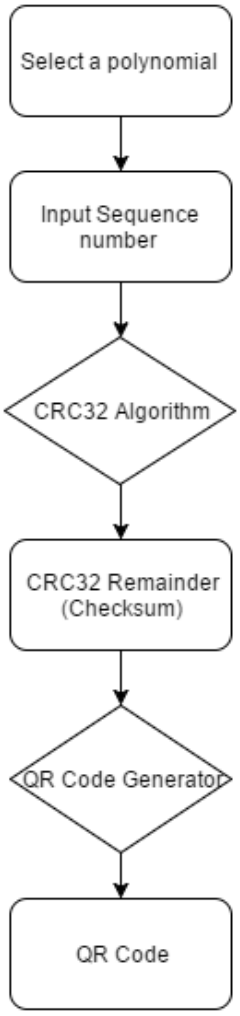

Figure 8.CRC32 Sender

On the sender side, a polynomial value, which is 0X04C11DB6 by default is reached. Then we input the product sequence number and based on a CRC32 algorithm function written in JAVASCRIPT we get the CRC32 remainder value, which is the checksum. In the last step, a QR code which will be printed on the products needs to be generated. First a combination of the sequence number is made, the checksum and the server information and the result will be

a URL, then this URL is sent to the QR code generation function and hence a QR code is generated.

3.3.2 CRC32 Receiver

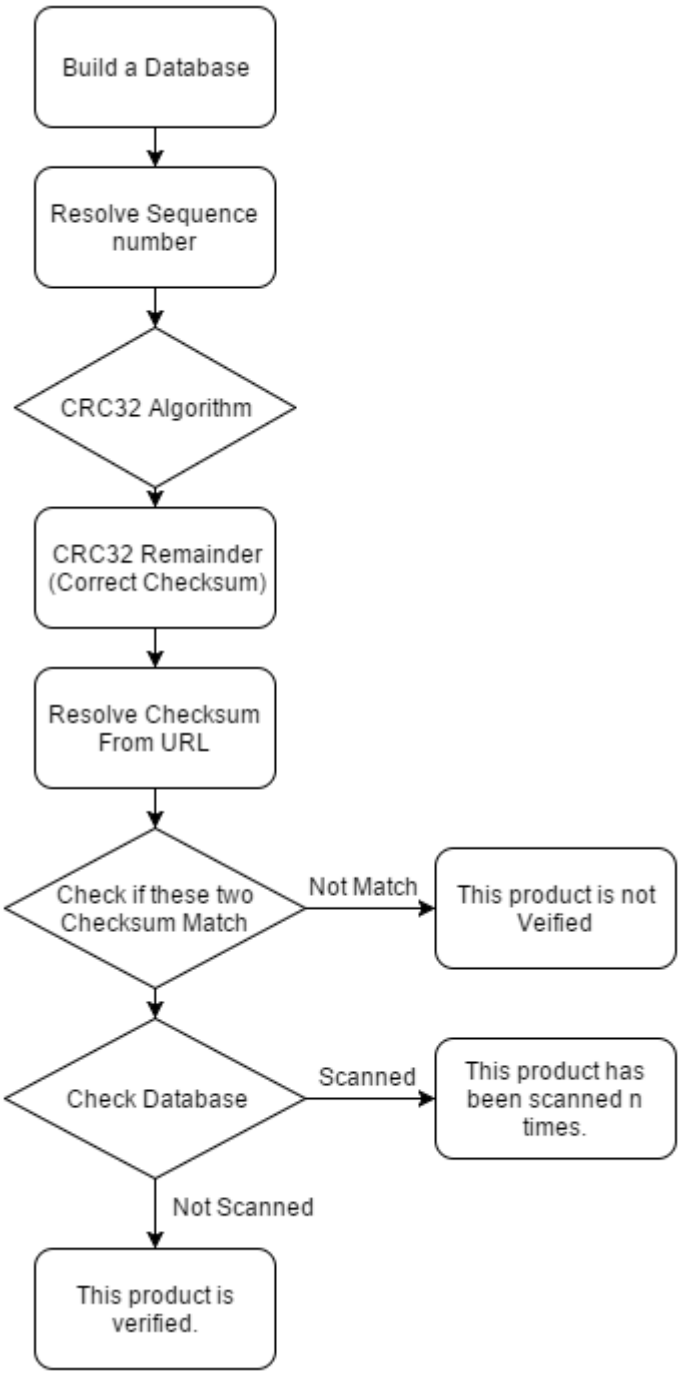

Figure 9.CRC32 Receiver

On the receiver side, a database is built which stores all the sequence numbers that have been scanned before. Then the sequence number is resolved and checksum information from the URL are obtained and the correct checksum value is calculated based on the sequence number by another CRC32 algorithm function written in PHP. The checking part is a server-side solution so that the algorithm and polynomial is hidden from potential hackers. After getting the correct checksum value, it is checked if the checksum value obtained from the URL matches the correct checksum value. If they do not match, this product is not verified. If they match, we refer to the database and check whether it has been scanned before, if it has, we return the scan time. If not, this product is verified.

4 TEST AND RESULT

With the principles introduced in Chapter 2 and the implementation methods introduced in Chapter 3, this project works successfully. In this chapter, the detailed implementation results are displayed.

4.1 CRC Sender Test

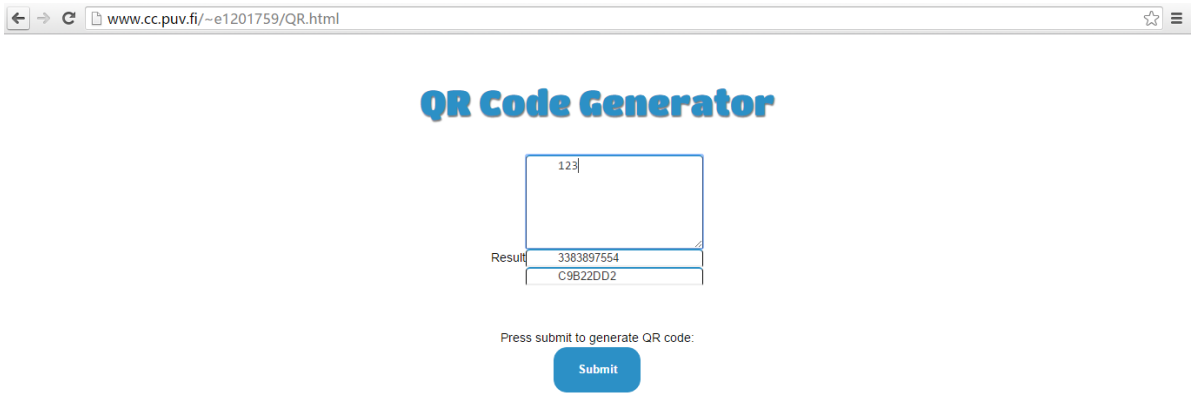

Figure 10.CRC Sender Test

Figure 10 shows how the CRC sender works in this project. On the data input area, the product sequence number is input, 123 is used as an example. Once the sequence number is entered, a CRC32 algorithm function is called and the result 3383897554 is displayed. This is the CRC32 remainder value. It can be also expressed as the hex decimal format 0XC9B22DD2. After generating the remainder value, once the submit button is clicked, a URL which contains the sequence number and the remainder value is sent to the QR code generation function thus a QR code is generated.

4.2 CRC Receiver Test

When a QR code is scanned on a product, the sequence number from the URL is resolved and the correct checksum value is calculated. Then the checksum number from the URL is resolved and check if this value matches the correct checksum value. If they do not match, this product is not verified. If they match, the server will check the database (Figure 11) whether it contains this sequence number and if it is stored, then the scan time is returned, if not, this product is verified.

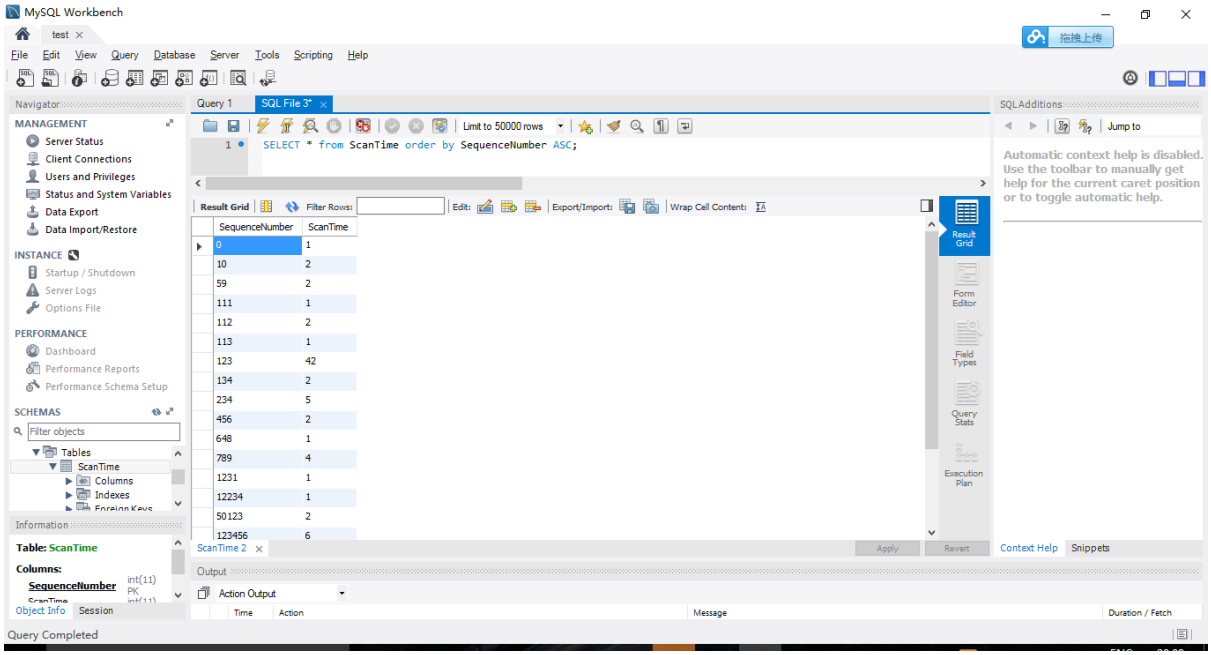

Figure 11.MySQL Database

Figure 12.CRC Receiver Test 1

Figure 12 shows an example when the product's checksum value does not match the correct one.

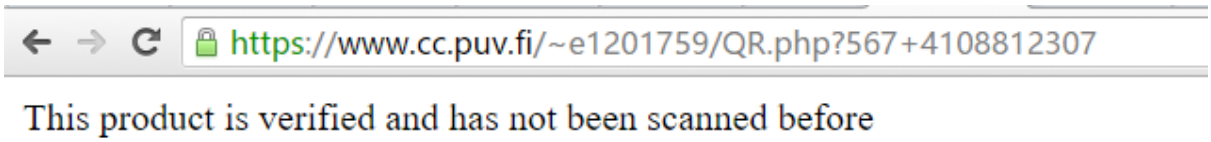

Figure 13.CRC Receiver Test 2

Figure 13 shows an example when the product's checksum value matches and the QR code is scanned for the first time.

This product has been scanned for 1 times

Figure 14.CRC Receiver Test 3

Figure 14 shows an example when the product's checksum value matches and the QR code has been scanned before.

4.3 Summary

Each product has a sequence number, therefore, a checksum value can be generated according to this sequence number on both the sender side and receiver side. Also the product is verified only when the checksum values match each other and what has not been scanned before.

Since each product has a unique sequence number, the checksum value for each sequence number is also unique. Therefore, there is low risk that two sequence numbers share a common checksum value. In addition, the CRC32 algorithm applies a 32-bits polynomial and therefore it will be difficult for hackers to figure out the polynomial value, hence, guaranteeing the information security.

5 CONCLUSION

Overall, this project introduces a way to facilitate the process of verifying a product. Without any need to get access to a specific device or dial a phone number, it requires only a smart phone installed with a QR code scanner application.

In addition, there is almost no additional cost required to implement this project. In spite of simply generating a QR code on the website and printing it on the surface of the product, all that needs to be done is to maintain a database.

The verification system is based on CRC32 algorithm. A QR code containing the product sequence number and the checksum value is generated on the sender side that on the receiver side, we need to generate a checksum value as well to check whether the checksum value matches. A product is verified only when the checksum value matches and the sequence number is not stored in the database.

During this project I have had a basic command of QR code generation technology and learnt how a CRC32 algorithm works. It also enhances my programming skills with JAVASCRIPT and PHP.

REFERENCE

/1/ Github QR Code Library. Accessed 30.3.2016

https://github.com/davidshimjs/qrcodejs

/2/ QR Code Definition. Accessed 27.3.2016

https://en.wikipedia.org/wiki/QR_code

/3/ CRC Definition. Accessed 15.4.2016

https://en.wikipedia.org/wiki/Cyclic_redundancy_check#Designing_polynomials

/4/ Ross N. Williams., 19 August 1993. A PAINLESS GUIDE TO CRC ERROR DETECTION ALGORITHMS. Accessed 15.4.2016

http://www.zlib.net/crc_v3.txt

/5/ ISO/IEC Standard 18004, "Information technology - Automatic identification and data capture techniques - Bar code symbology - QR Code", ISO/IEC, 2000. Accessed 15.4.2016

http://www.swisseduc.ch/informatik/theoretische_informatik/qr_codes/docs/qr_standard.pdf

/6/ QR Code Module Accessed 18.5.2016

http://www.qrcode.com/en/howto/cell.html

APPENDIX 1. CRC Sender Source Code

```
 <html>

<head></head>

<body>

<script type="text/javascript" src="./js/jquery.min.js"></script>

<script type="text/javascript" src="./js/jquery.qrcode.js"></script> /1/

<script type="text/javascript" src="./js/qrcode.js"></script>

<link href="./css/test3.css" rel='stylesheet' type='text/css' />

<script type="text/javascript">

/*The following table stores all the crc32 remainder(checksum) for the polynomial
value 6DB88320(reversed polynomial of 04C11DB6) */

var table =
       [0x00000000,0x36073096,0x6C0E612C,0x5A0951BA,0x036DC419,0x356AF48F,
       0x6F63A535,0x596495A3,0x06DB8832,0x30DCB8A4,0x6AD5E91E,0x5CD2D988,
       0x05B64C2B,0x33B17CBD,0x69B82D07,0x5FBF1D91,0x0DB71064,0x3BB020F2,
       0x61B97148,0x57BE41DE,0x0EDAD47D,0x38DDE4EB,0x62D4B551,0x54D385C7,
       0x0B6C9856,0x3D6BA8C0,0x6762F97A,0x5165C9EC,0x08015C4F,0x3E066CD9,

       0x640F3D63,0x52080DF5,0x1B6E20C8,0x2D69105E,0x776041E4,0x41677172,
       0x1803E4D1,0x2E04D447,0x740D85FD,0x420AB56B,0x1DB5A8FA,0x2BB2986C,
       0x71BBC9D6,0x47BCF940,0x1ED86CE3,0x28DF5C75,0x72D60DCF,0x44D13D59,
       0x16D930AC,0x20DE003A,0x7AD75180,0x4CD06116,0x15B4F4B5,0x23B3C423,
       0x79BA9599,0x4FBDA50F,0x1002B89E,0x26058808,0x7C0CD9B2,0x4A0BE924,
       0x136F7C87,0x25684C11,0x7F611DAB,0x49662D3D,0x36DC4190,0x00DB7106,
       0x5AD220BC,0x6CD5102A,0x35B18589,0x03B6B51F,0x59BFE4A5,0x6FB8D433,
       0x3007C9A2,0x0600F934,0x5C09A88E,0x6A0E9818,0x336A0DBB,0x056D3D2D,
       0x5F646C97,0x69635C01,0x3B6B51F4,0x0D6C6162,0x576530D8,0x6162004E,
       0x380695ED,0x0E01A57B,0x5408F4C1,0x620FC457,0x3DB0D9C6,0x0BB7E950,
       0x51BEB8EA,0x67B9887C,0x3EDD1DDF,0x08DA2D49,0x52D37CF3,0x64D44C65,
       0x2DB26158,0x1BB551CE,0x41BC0074,0x77BB30E2,0x2EDFA541,0x18D895D7,
       0x42D1C46D,0x74D6F4FB,0x2B69E96A,0x1D6ED9FC,0x47678846,0x7160B8D0,
       0x28042D73,0x1E031DE5,0x440A4C5F,0x720D7CC9,0x2005713C,0x160241AA,
       0x4C0B1010,0x7A0C2086,0x2368B525,0x156F85B3,0x4F66D409,0x7961E49F,
       0x26DEF90E,0x10D9C998,0x4AD09822,0x7CD7A8B4,0x25B33D17,0x13B40D81,

       0x49BD5C3B,0x7FBA6CAD,0x6DB88320,0x5BBFB3B6,0x01B6E20C,0x37B1D29A,
       0x6ED54739,0x58D277AF,0x02DB2615,0x34DC1683,0x6B630B12,0x5D643B84,
       0x076D6A3E,0x316A5AA8,0x680ECF0B,0x5E09FF9D,0x0400AE27,0x32079EB1,
       0x600F9344,0x5608A3D2,0x0C01F268,0x3A06C2FE,0x6362575D,0x556567CB,
       0x0F6C3671,0x396B06E7,0x66D41B76,0x50D32BE0,0x0ADA7A5A,0x3CDD4ACC,
       0x65B9DF6F,0x53BEEFF9,0x09B7BE43,0x3FB08ED5,0x76D6A3E8,0x40D1937E,
       0x1AD8C2C4,0x2CDFF252,0x75BB67F1,0x43BC5767,0x19B506DD,0x2FB2364B,
       0x700D2BDA,0x460A1B4C,0x1C034AF6,0x2A047A60,0x7360EFC3,0x4567DF55,
       0x1F6E8EEF,0x2969BE79,0x7B61B38C,0x4D66831A,0x176FD2A0,0x2168E236,
       0x780C7795,0x4E0B4703,0x140216B9,0x2205262F,0x7DBA3BBE,0x4BBD0B28,
       0x11B45A92,0x27B36A04,0x7ED7FFA7,0x48D0CF31,0x12D99E8B,0x24DEAE1D,
```

```
	0x5B64C2B0,0x6D63F226,0x376AA39C,0x016D930A,0x580906A9,0x6E0E363F,
	0x34076785,0x02005713,0x5DBF4A82,0x6BB87A14,0x31B12BAE,0x07B61B38,
	0x5ED28E9B,0x68D5BE0D,0x32DCEFB7,0x04DBDF21,0x56D3D2D4,0x60D4E242,
	0x3ADDB3F8,0x0CDA836E,0x55BE16CD,0x63B9265B,0x39B077E1,0x0FB74777,

	0x50085AE6,0x660F6A70,0x3C063BCA,0x0A010B5C,0x53659EFF,0x6562AE69,
	0x3F6BFFD3,0x096CCF45,0x400AE278,0x760DD2EE,0x2C048354,0x1A03B3C2,
	0x43672661,0x756016F7,0x2F69474D,0x196E77DB,0x46D16A4A,0x70D65ADC,
	0x2ADF0B66,0x1CD83BF0,0x45BCAE53,0x73BB9EC5,0x29B2CF7F,0x1FB5FFE9,
	0x4DBDF21C,0x7BBAC28A,0x21B39330,0x17B4A3A6,0x4ED03605,0x78D70693,
	0x22DE5729,0x14D967BF,0x4B667A2E,0x7D614AB8,0x27681B02,0x116F2B94,

	0x480BBE37,0x7E0C8EA1,0x2405DF1B,0x1202EF8D];

	function crc32_calculate(str) {

	/*Calculate the checksum according to the input value*/

		var crc = 0xFFFFFFFF //Set CRC Original Value

		var i

		for (i = 0; i < str.length; i++)

		//Refer to the table and calculate the remainder for one byte

		crc = (crc >>> 8) ^ table[str.charCodeAt(i) ^ (crc & 0x000000FF)]

		crc = ~crc //Reverse the result

		if (crc < 0)

			{

			crc = 0xFFFFFFFF + crc + 1;//Calculate complement if it is
negative

			}

		return crc

	}

</script>

<p class="texto">QR Code Generator</p>

<form name=form1>

    <table class=wide>

        <tr>

            <td valign=top></td>
```

```
            <td class=wide><textarea name=data type=text id=data placeholder="Plese
input       data       here"       autofocus       rows=5       onChange="onChange()"
onKeyUp="onChange()"></textarea></td>

        </tr>

        <tr>

            <td>Result</td>

            <td><input type=text name=decResult id=decResult readonly> </td>

        </tr>

        <tr>

            <td> </td>

            <td><input type=text name=hexResult id=hexResult readonly> </td>

        </tr>

    </table>

</form>

<div id="qrcodeTable"></div>

<p>Press submit to generate QR code:</p>

<button class="button" onclick="submit();" href="javascript:;">Submit</button>

<script type="text/javascript">

   String.prototype.repeat = function(times) {

       return (new Array(times + 1)).join(this);

    }

    Number.prototype.toHex = function(len)

      /*Convert decimal to hex and add padding zero if the length is less than 8*/

      {

        if (typeof(len) === 'undefined') len = 8;

        var num = this < 0 ? (0xFFFFFFFF + this + 1) : this

        var hex = num.toString(16).toUpperCase()

        var pad = hex.length < len ? len - hex.length : 0

        return "0".repeat(pad) + hex;
```

```
    }

       function onChange() {

/*Update the result once the data changes*/

                    var data = document.form1.data.value

                    var crc = crc32_calculate(data);

                    document.form1.decResult.value = crc

                    document.form1.hexResult.value = crc.toHex()

             }

       function submit(){

                    $('canvas').remove();

                    //var                          reversed                          =
parseInt(document.form1.polynomialReversed.value, 16)

                    var data = document.form1.data.value

                    var crc = document.form1.decResult.value.toString()

                    //alert(crc);
jQuery('#qrcodeTable').qrcode("https://www.cc.puv.fi/~e1201759/QR.php?"+data+"+"+c
rc);

         }

</script>

</body>

</html>
```

APPENDIX 2. CRC Receiver Source Code

```
 <!DOCTYPE html>
<html>
<body>
<?php
/* php crc32 implementation */
function zeroFill($a, $b) {
/*shift one bit and add padding zero*/
	$z = hexdec(80000000);
	$z = (int)$z;
	$a = (int)$a;
	$b = (int)$b;
	if ($z & $a) {
		$a >>= 1;
		$a &= (~ $z);
		$a |= 0x40000000;
		$a >>= ($b-1);
	}
	else {
	       $a >>= $b;
	}
	return $a;
}
function crc32_generate($polynomial) {
	/*Pre-generate a table which stores all the remainder
	for numbers between 0 and 255*/
    $table = array();
    for ($i = 0; $i < 256; $i++) {
```

```
        $n = $i;
        for ($j = 8; $j > 0; $j--) {
            if (($n & 1) == 1) {
                $n = (zeroFill($n, 1)) ^ $polynomial;
            } else {
                $n = zeroFill($n, 1);
            }
        }
        $table[$i] = $n;
    }
    return $table;
}
function crc32_initial() {
       /*set crc32 initial value*/
    return 0xFFFFFFFF;
}
function crc32_final($crc) {
       /*reverse the result and use complement if the value is negative*/
    $crc = ~$crc;
    return $crc < 0 ? 0xFFFFFFFF + $crc + 1 : $crc;
}
function crc32_compute_string($polynomial, $str) {
       /*compute the checksum according to the sequence number*/
    $crc = 0;
    $table = crc32_generate($polynomial);

    $crc = crc32_initial();
```

```
    for ($i = 0; $i < strlen($str); $i++)
        $crc = (zeroFill($crc, 8)) ^ $table[ord($str[$i]) ^ ($crc & 0x000000FF)];
    $crc = crc32_final($crc);
    return $crc;
}
function crc32_reversed($polynomial) {
       /*reverse polynomial value*/
    $reversed = 0;
    for ($i = 0; $i < 32; $i++) {
        $reversed = $reversed << 1;
        $reversed = $reversed | (zeroFill($polynomial, $i) & 1);
    }
    return reversed;
}

$polynomialReversed="6DB88320";
$result= $_SERVER["QUERY_STRING"]."<br>";
$result1=explode("+",$result);
$result2=$result1[1];
$result3=$result1[0];
$str=crc32_compute_string(hexdec($polynomialReversed),$result1[0]);
$servername = "mysql.cc.puv.fi";
$username = "e1201759";
$password = "";
$dbname = "e1201759_SequenceNumber";
// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
//sql to create table
```

```
$sql = "CREATE TABLE IF NOT EXISTS ScanTime(

SequenceNumber INT NULL,

ScanTime INT NULL,

PRIMARY KEY (`SequenceNumber`)

)";

$conn->query($sql);

//search for the sequence number in the database

$sql = "SELECT SequenceNumber,ScanTime FROM ScanTime

where SequenceNumber='$result3'";

$result4 = $conn->query($sql);

if($str==$result2 && $str!=""){

	if ($result4->num_rows > 0) {

		$row=mysqli_fetch_assoc($result4);

		printf	("This	product	has	been	scanned	for	%s
times\n",$row["ScanTime"]);

		$sql = "UPDATE ScanTime SET ScanTime=ScanTime+1

		WHERE SequenceNumber='$result3'";

		$conn->query($sql);

	}

	else {

		$sql = "INSERT INTO ScanTime (SequenceNumber, ScanTime)

		VALUES ('$result3',1)";

		$conn->query($sql);

		echo "This product is verified and has not been scanned before";

	}

}

else{

	echo "This product is not verified";

}
```

```
?>

</body>

</html>
```

APPENDIX 3. CRC Table Generation Source Code

```
<html>
<body>
<script type="text/javascript">
function crc32_generate(polynomial) {
/*This function is used to generate crc32 remainder value for numbers between
0-255 based on the polynomial value 0x6DB88320.*/
    polynomial = 0x6DB88320
    var table = new Array()
    var i, j, n

    for (i = 0; i < 256; i++) {
        n = i
        for (j = 8; j > 0; j--) {
            if ((n & 1) == 1) {
                n = (n >>> 1) ^ polynomial
            } else {
                n = n >>> 1
            }
        }
        table[i] = n
    }
       document.getElementById("demo").innerHTML = table;
    return table
}
</script>
</body>
</html>
```