Base64

Computer Security Assignment 01

202004520 최준혁

What is Base64

(From MDN)

Base64 is a group of similar binary-to-text encoding schemes that represent binary data in an ASCII string format by transforming it into a radix-64 representation.

Simply, Base64 is one of a way to represent binary data* as ASCII

* Texts are also “binary data”

So base64 encoded data only contains alphabet, number, and special characters ‘+’, ‘/’. (sometimes contains ‘=’)

Why we need Base64

Base64: Binary data <-> ASCII

We can encode almost every computer-based data to Base64

and can represent Base64 strings in almost every computer devices (because most computers using strings as ASCII)

Example: Unicode Characters

Unicode ASCII

Sometimes application, interface, devices (or etc.) only accept ASCII characters.
In this case, you can send unicode characters* by encoding to base64

* Unicode characters: Non-ASCII characters. Generally encoded into UTF-8
(example: 한글(Korean), 日本語(Japanese) and etc)

Example: HTTP

Despite HTTP can process binary data, sometimes we need to send binary data as “plain text”.
(ex: multipart/form-data)

In this case, we can use MIME types and Base64.
As you know, Base64 will encode binary data to ASCII texts. (and ASCII is “plain text”.)
And MIME types represent type of data.
(Example of MIME type: “image/jpeg”, “video/mp4”)

By receiver decoding b64 strings to binary data and recognize this data with MIME types,
It can use binary data correctly.

Example: HTTP

Example of multipart/form-data raw HTTP request

```
POST /upload HTTP/1.1
Host: example.com
Content-Type: multipart/form-data; boundary=----WebKitFormBoundary7MA4YWxkTrZu0gW
Content-Length: 111111

------WebKitFormBoundary7MA4YWxkTrZu0gW
Content-Disposition: form-data; name="text_field"

This is a text field
------WebKitFormBoundary7MA4YWxkTrZu0gW
Content-Disposition: form-data; name="file"; filename="example.txt"
Content-Type: text/plain
Content-Transfer-Encoding: base64

SGVsbG8gd29ybGQhCg==
------WebKitFormBoundary7MA4YWxkTrZu0gW--
```

Careful point of Base64

Base64 is actually not a “crypto” algorithm.

As I explained above, Base64 was designed for convert binary data to ASCII (plain text)

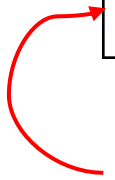
Of course, human can’t read (or need to spent many time) b64 encoded data

But it’s not safe because it using static and opened mapping table.

How base64 implemented

1. Get bit pattern to convert.
In this case, we convert ASCII texts.

Text	H								E								L								L								O							
ASCII	72								69								76								76								79							
bit	0	1	0	0	1	0	0	0	0	1	0	0	0	1	0	1	0	1	0	0	1	1	0	0	0	1	0	0	1	1	0	0	0	1	0	0	1	1	1	1



We need this one

How base64 implemented

2. Slice bit pattern into 6 bit (so, 2^6 = 64 digit)
If it's not enough for 6 bit, add 0 as padding

Text	H								E								L								L								O									
ASCII	72								69								76								76								79									
bit	0	1	0	0	1	0	0	0	0	1	0	0	0	1	0	1	0	1	0	0	1	1	0	0	0	1	0	0	1	1	0	0	0	1	0	0	1	1	1	1	0	0
Sliced	**0**	**1**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**1**	**0**	**1**	**0**	**1**	**0**	**0**	**1**	**1**	**0**	**0**	**0**	**1**	**0**	**0**	**1**	**1**	**0**	**0**	**0**	**1**	**0**	**0**	**1**	**1**	**1**	**1**	**0**	**0**

Add padding bit

How base64 implemented

3. Convert sliced bit to decimal

Text	H								E								L								L								O									
ASCII	72								69								76								76								79									
bit	0	1	0	0	1	0	0	0	0	1	0	0	0	1	0	1	0	1	0	0	1	1	0	0	0	1	0	0	1	1	0	0	0	1	0	0	1	1	1	1	0	0
Sliced	**0**	**1**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**1**	**0**	**1**	**0**	**1**	**0**	**0**	**1**	**1**	**0**	**0**	**0**	**1**	**0**	**0**	**1**	**1**	**0**	**0**	**0**	**1**	**0**	**0**	**1**	**1**	**1**	**1**	**0**	**0**
dec	**18**								**4**								**21**								**12**								**19**								**4**				**60**			

How base64 implemented

BASE64 INDEX TABLE

0	A	16	Q	32	g	48	w
1	B	17	R	33	h	49	x
2	C	18	S	34	i	50	y
3	D	19	T	35	j	51	z
4	E	20	U	36	k	52	0
5	F	21	V	37	l	53	1
6	G	22	W	38	m	54	2
7	H	23	X	39	n	55	3
8	I	24	Y	40	o	56	4
9	J	25	Z	41	p	57	5
10	K	26	a	42	q	58	6
11	L	27	b	43	r	59	7
12	M	28	c	44	s	60	8
13	N	29	d	45	t	61	9
14	O	30	e	46	u	62	+
15	P	31	f	47	v	63	/

4. Convert decimal to base64 string from base64 table*
* Sliced to 6-bit so it's value will be 0~63.
0 will be 'A' and 26 will be 'a'. 52~61 will be '0'~'9'.
Remained 62 and 63 will be '+' and '/'

Text	H								E								L								L								O									
ASCII	72								69								76								76								79									
bit	0	1	0	0	1	0	0	0	0	1	0	0	0	1	0	1	0	1	0	0	1	1	0	0	0	1	0	0	1	1	0	0	0	1	0	0	1	1	1	1	0	0
Sliced	**0**	**1**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**1**	**0**	**1**	**0**	**1**	**0**	**0**	**1**	**1**	**0**	**0**	**0**	**1**	**0**	**0**	**1**	**1**	**0**	**0**	**0**	**1**	**0**	**0**	**1**	**1**	**1**	**1**	**0**	**0**
dec	**18**								**4**								**21**								**12**								**19**								**4**								**60**							
b64	**S**								**E**								**V**								**M**								**T**								**E**								**8**							

Result: SEVMTE8

How base64 implemented

Sometimes add '=' as padding for matching 8-bytes.
(slicing b64 string into 4 characters and add '=' behind)

Text	H								E								L								L								O									
ASCII	72								69								76								76								79									
bit	0	1	0	0	1	0	0	0	0	1	0	0	0	1	0	1	0	1	0	0	1	1	0	0	0	1	0	0	1	1	0	0	0	1	0	0	1	1	1	1	0	0
Sliced	**0**	**1**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**1**	**0**	**1**	**0**	**1**	**0**	**0**	**1**	**1**	**0**	**0**	**0**	**1**	**0**	**0**	**1**	**1**	**0**	**0**	**0**	**1**	**0**	**0**	**1**	**1**	**1**	**1**	**0**	**0**
dec	**18**								**4**								**21**								**12**								**19**								**4**								**60**							
b64	**S**								**E**								**V**								**M**								**T**								**E**								**8**							

Result b64 string is "SEVMTE8" and it's 7 letter.
So add '=' behind
Result: SEVMTE8=

How base64 implemented

And of course, we can convert any data if it is binary data.
Example: UTF-8 Non-ASCII Text

Text	안																녕															
UTF-8 hex	EC								95								88								EB							
bit	1	1	1	0	1	1	0	0	1	0	0	1	0	1	0	1	1	0	0	0	1	0	0	0	1	1	1	0	1	0	1	1
sliced	1	1	1	0	1	1	0	0	1	0	0	1	0	1	0	1	1	0	0	0	1	0	0	0	1	1	1	0	1	0	1	1
dec	59								9				22				8				58				56							
b64	7								J				W				I				6				4							

1	1	1	0	1	1	0	0	1	0	0	1	0	1	0	1	1	0	0	0	1	0	0	0	1	1	1	0	1	0	1	1	1	0	0	0	0	1	0	1	1	0	0	1	0	1	0	1
1	1	1	0	1	1	0	0	1	0	0	1	0	1	0	1	1	0	0	0	1	0	0	0	1	1	1	0	1	0	1	1	1	0	0	0	0	1	0	1	1	0	0	1	0	1	0	1
59								9				22				8				58				56				22				21			
7								J				W				I				6				4				W				V			

Result: 7JWI64WV

Implementation Source code (Node.js)

HUFS.CS.2024-02/assignment/01 at main · RFLXN/HUFS.CS.2024-02 (github.com)