1. /\*\*
2. Aes encryption
3. \*/
4. **class** AES {
5. **const** M\_CBC = 'cbc';
6. **const** M\_CFB = 'cfb';
7. **const** M\_ECB = 'ecb';
8. **const** M\_NOFB = 'nofb';
9. **const** M\_OFB = 'ofb';
10. **const** M\_STREAM = 'stream';
11. protected **$key**;
12. protected **$cipher**;
13. protected **$data**;
14. protected **$mode**;
15. protected **$IV**;
16. /\*\*
17. \*
18. \* **@param** type $data
19. \* **@param** type $key
20. \* **@param** type $blockSize
21. \* **@param** type $mode
22. \*/
23. **function** \_\_construct(**$data** = null, **$key** = null, **$blockSize** = null, **$mode** = null) {
24. **$this**->setData(**$data**);
25. **$this**->setKey(**$key**);
26. **$this**->setBlockSize(**$blockSize**);
27. **$this**->setMode(**$mode**);
28. **$this**->setIV("");
29. }
30. /\*\*
31. \*
32. \* **@param** type $data
33. \*/
34. public **function** setData(**$data**) {
35. **$this**->data = **$data**;
36. }
37. /\*\*
38. \*
39. \* **@param** type $key
40. \*/
41. public **function** setKey(**$key**) {
42. **$this**->key = **$key**;
43. }
44. /\*\*
45. \*
46. \* **@param** type $blockSize
47. \*/
48. public **function** setBlockSize(**$blockSize**) {
49. **switch** (**$blockSize**) {
50. **case** 128:
51. **$this**->cipher = MCRYPT\_RIJNDAEL\_128;
52. **break**;
53. **case** 192:
54. **$this**->cipher = MCRYPT\_RIJNDAEL\_192;
55. **break**;
56. **case** 256:
57. **$this**->cipher = MCRYPT\_RIJNDAEL\_256;
58. **break**;
59. }
60. }
61. /\*\*
62. \*
63. \* **@param** type $mode
64. \*/
65. public **function** setMode(**$mode**) {
66. **switch** (**$mode**) {
67. **case** AES::M\_CBC:
68. **$this**->mode = MCRYPT\_MODE\_CBC;
69. **break**;
70. **case** AES::M\_CFB:
71. **$this**->mode = MCRYPT\_MODE\_CFB;
72. **break**;
73. **case** AES::M\_ECB:
74. **$this**->mode = MCRYPT\_MODE\_ECB;
75. **break**;
76. **case** AES::M\_NOFB:
77. **$this**->mode = MCRYPT\_MODE\_NOFB;
78. **break**;
79. **case** AES::M\_OFB:
80. **$this**->mode = MCRYPT\_MODE\_OFB;
81. **break**;
82. **case** AES::M\_STREAM:
83. **$this**->mode = MCRYPT\_MODE\_STREAM;
84. **break**;
85. **default**:
86. **$this**->mode = MCRYPT\_MODE\_ECB;
87. **break**;
88. }
89. }
90. /\*\*
91. \*
92. \* **@return** boolean
93. \*/
94. public **function** validateParams() {
95. **if** (**$this**->data != null &&
96. **$this**->key != null &&
97. **$this**->cipher != null) {
98. **return** true;
99. } **else** {
100. **return** FALSE;
101. }
102. }
103. public **function** setIV(**$IV**) {
104. **$this**->IV = **$IV**;
105. }
106. protected **function** getIV() {
107. **if** (**$this**->IV == "") {
108. **$this**->IV = mcrypt\_create\_iv(mcrypt\_get\_iv\_size(**$this**->cipher, **$this**->mode), MCRYPT\_RAND);
109. }
110. **return** **$this**->IV;
111. }
112. /\*\*
113. \* **@return** type
114. \* **@throws** Exception
115. \*/
116. public **function** encrypt() {
117. **if** (**$this**->validateParams()) {
118. **return** trim(base64\_encode(
119. mcrypt\_encrypt(
120. **$this**->cipher, **$this**->key, **$this**->data, **$this**->mode, **$this**->getIV())));
121. } **else** {
122. throw **new** Exception('Invlid params!');
123. }
124. }
125. /\*\*
126. \*
127. \* **@return** type
128. \* **@throws** Exception
129. \*/
130. public **function** decrypt() {
131. **if** (**$this**->validateParams()) {
132. **return** trim(mcrypt\_decrypt(
133. **$this**->cipher, **$this**->key, base64\_decode(**$this**->data), **$this**->mode, **$this**->getIV()));
134. } **else** {
135. throw **new** Exception('Invlid params!');
136. }
137. }
138. }