

Feb 27, 19 8:51

crypto.rs

Page 1/2

```

1  /*
2  *      Crypto for Bitwarden against openssl.
3  *
4  */
5  extern crate hmac;
6  extern crate openssl;
7  extern crate rustc_serialize as serialize;
8  use serialize::base64;
9  // use serialize::base64::FromBase64;
10 use serialize::base64::ToBase64;
11 // use std::io;
12
13 pub fn hmac_openssl(key: &[u8], messages: &str) -> String {
14     let pkey = openssl::pkey::PKey::hmac(&key).unwrap();
15     let mut signer = openssl::sign::Signer::new(openssl::hash::MessageDigest::sha256(), &pkey).unwrap();
16     signer.update(messages.as_bytes()).unwrap();
17     let mut result = String::from("");
18     result.push_str(&signer.sign_to_vec().unwrap().to_base64(base64::STANDARD)[..]);
19     return result;
20 }
21
22 //make a master key.
23 pub fn make_key(password: &str, salt: &str) -> [u8; 32] {
24     // 256-bit derived key
25     // hashlib.pbkdf2_hmac('sha256', password, salt, 5000, dklen=32)
26     //let mut dk = [0u8; 32];
27     //let mut dk = [0u8; 32];
28     let mut derived_key = [0; 32];
29     openssl::pkcs5::pbkdf2_hmac(&password.as_bytes(), &salt.as_bytes(), 5000, openssl::hash::MessageDigest::sha256(), &mut derived_key).unwrap();
30     //let mut result = String::from("");
31     //result.push_str(&derived_key.to_base64(base64::STANDARD)[..]);
32     return derived_key;
33 }
34
35 ///# base64-encode a wrapped, stretched password+salt for signup/login
36 pub fn hashed_password(password: &str, salt: &str) -> String {
37     let key = make_key(password, salt);
38     //let mut derived_key = [0u8; 32];
39     let mut derived_key = [0; 32];
40     openssl::pkcs5::pbkdf2_hmac(&key, &password.as_bytes(), 1, openssl::hash::MessageDigest::sha256(), &mut derived_key).unwrap();
41     let mut result = String::from("");
42     result.push_str(&derived_key.to_base64(base64::STANDARD)[..]);
43     return result;
44 }
45
46 // encode into a bitwarden compatible cipher string.
47 pub fn encode_cipher_string(enctype: &u8, iv: &[u8], ct: &[u8], mac: &[u8]) -> String {
48     let mut result = String::from("");
49     result.push_str(&enctype.to_string());
50     result.push_str(".");
51     if iv.len() > 0 {
52         result.push_str(&iv.to_base64(base64::STANDARD)[..]);
53     }
54     if ct.len() > 0 {
55         result.push_str("|");
56         result.push_str(&ct.to_base64(base64::STANDARD)[..]);
57     }
58     if mac.len() > 0 {
59         result.push_str("|");
60         result.push_str(&mac.to_base64(base64::STANDARD)[..]);
61     }
62     return result;
63 }
64 pub struct Cipherstring {
65     encryption_type: u8,
66     iv: Vec<u8>,
67     ct: Vec<u8>,
68     mac: Vec<u8>,
69 }
70
71 //decode a bitwarden cipher string
72 pub fn decode_cipher_string(cipher_string: &str) -> Cipherstring {
73     let pieces: Vec<&str> = cipher_string.split("|").collect();
74     let beg = pieces[0];
75     let beg_pieces: Vec<&str> = beg.split(".").collect();
76     let enc_type = beg_pieces[0];
77     let iv = beg_pieces[1];
78     let ct = pieces[1];
79     let mut mac = vec![0; 0];
80     if pieces.len() == 3 {
81         mac = pieces[2].from_base64().unwrap();
82     } else {
83         mac = [0; 0].to_vec();
84     }

```

Feb 27, 19 8:51

crypto.rs

Page 2/2

```

85     let result = Cipherstring {
86         encryption_type: enc_type.parse::<u8>().unwrap(),
87         iv: iv.from_base64().unwrap(),
88         ct: ct.from_base64().unwrap(),
89         mac,
90     };
91     return result;
92 }
93
94 //create symmetric key (encryption_key and mac_key from secure random bytes
95 // we return it as one large 64 byte variable, the first 32 are the encryption key and the last 32 are the mac_
96 key.
97 pub fn symmetric_key() -> (Vec<u8>) {
98     let mut rng = rand::thread_rng();
99     let encryption_key: Vec<u8> = rand::seq::sample_iter(&mut rng, 0..u8::max_value(), 32).unwrap();
100     let mac_key: Vec<u8> = rand::seq::sample_iter(&mut rng, 0..u8::max_value(), 32).unwrap();
101     return encryption_key + mac_key;
102 }
103
104 // make encryption key
105 pub fn make_encrypted_key(symmetric_key: Vec<u8>, master_key: [u8; 32]) -> String {
106     let mut rng = rand::thread_rng();
107     let iv: Vec<u8> = rand::seq::sample_iter(&mut rng, 0..u8::max_value(), 16).unwrap();
108     let cipher = encrypt_aes_256_cbc(&symmetric_key, &master_key, &iv).unwrap();
109     let mac: [u8; 0] = [];
110     let ret = encode_cipher_string(&0, &iv, &cipher, &mac);
111     return ret;
112 }
113
114 //double hmac compare.
115 pub fn macs_equal(mac_key: &[u8], mac1: &[u8], mac2: &[u8]) -> bool {
116     let mut hmac1 = hmac::Hmac::new(crypto::sha2::Sha256::new(), &mac_key);
117     hmac1.input(&mac1);
118     let mut hmac2 = hmac::Hmac::new(crypto::sha2::Sha256::new(), &mac_key);
119     hmac2.input(&mac2);
120     return hmac1.result() == hmac2.result();
121 }
122
123 #[cfg(test)]
124 mod tests {
125     use super::*;
126     #[test]
127     fn test_hashed_password() {
128         let result = hashed_password("password", "nobody@example.com");
129         let expected = "2cj6A0brDusMjVlVqcBW2a+kiOQDqZDCEB40NshJE7o=";
130         assert_eq!(expected, result);
131     }
132     #[test]
133     fn test_make_key() {
134         let expected = b"\x95\xa9\xc3\xb6W\xfb\xa7r\x80\xbfY\xdf\xfc\x18S\x81\x9e+\xf7W\xd0\x1db\x92$\x1bN\x05\x
135         xf5\xb8s\xe7";
136         let result = make_key("password", "nobody@example.com");
137         assert_eq!(expected, &result);
138     }
139     #[test]
140     fn test_decrypt_encrypted_key() {
141         let expected = b"";
142         //let result = decrypt_encrypted_key("0.QjjRqI96zTB7/z3wHInzg==|WHl3wQjcPmZJ4wgADXYwOhMB6RILrqPcivCJc5
143         00kivznCRaFTBXVe6MudXyCJEu6M7RMVQfz71LEcmcy/DFOT5veHR9YCdp4kQj3t4Tx0=",);
144         //assert_eq!(expected, &result);
145     }
146 }

```