Web Crypto API

end-to-end encryption & privacy


SERLI


DEVOXX
BELGIUM


SERLi

La crypto pour les ~~nuls~~ développeurs

Hachage

Fonction = Input => Digest

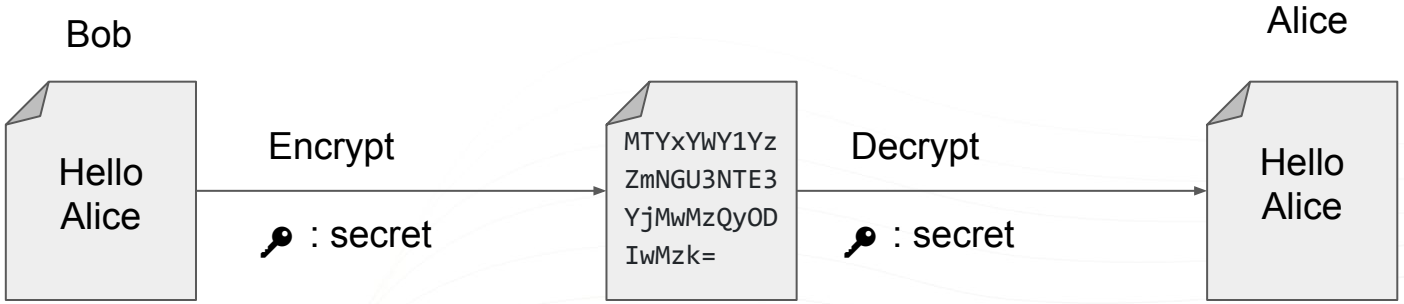
Identification unique (collision !!!), obfuscation des données,

dans le cas du hash de password, salage pour ajouter de l'entropie (si salt unique et aléatoire)

- Génération pseudo-aléatoire du salt
- Dérivation cryptographique sur [salt + password] (genre bcrypt)
- Stockage de tous les éléments en base
- Stockage d'une signature HMAC du payload

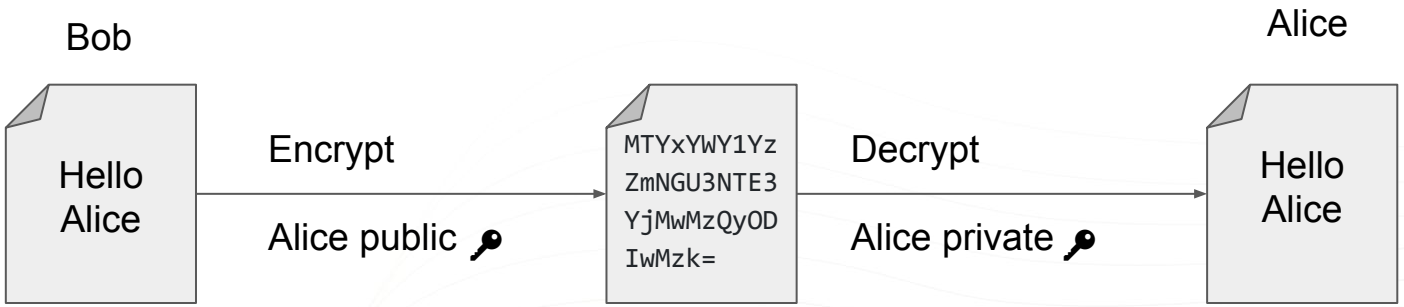
Chiffrement symétrique

Chiffrement basé sur une clé unique utilisée pour chiffrer et déchiffrer (AES, Blowfish, IDEA)



Chiffrement asymétrique

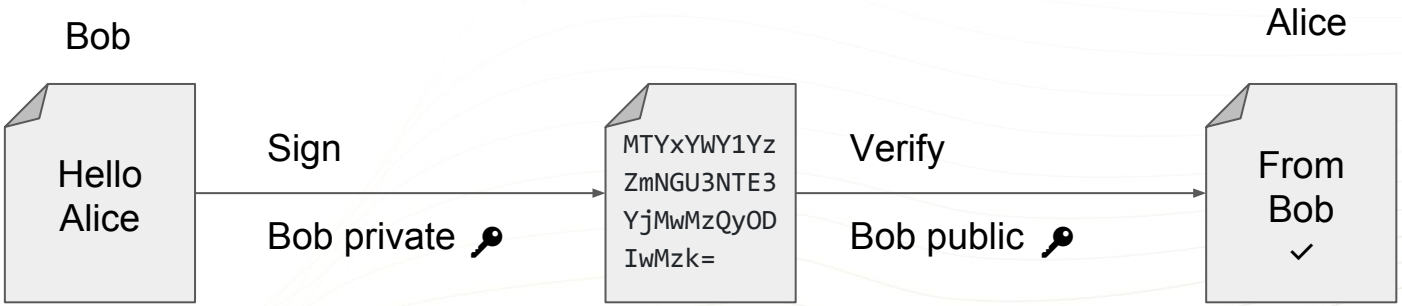
Chiffrement basé sur des paires de clés (publique / privée) (RSA)



Signature

Chiffrement asymétrique inversé

Permet de valider la provenance de données



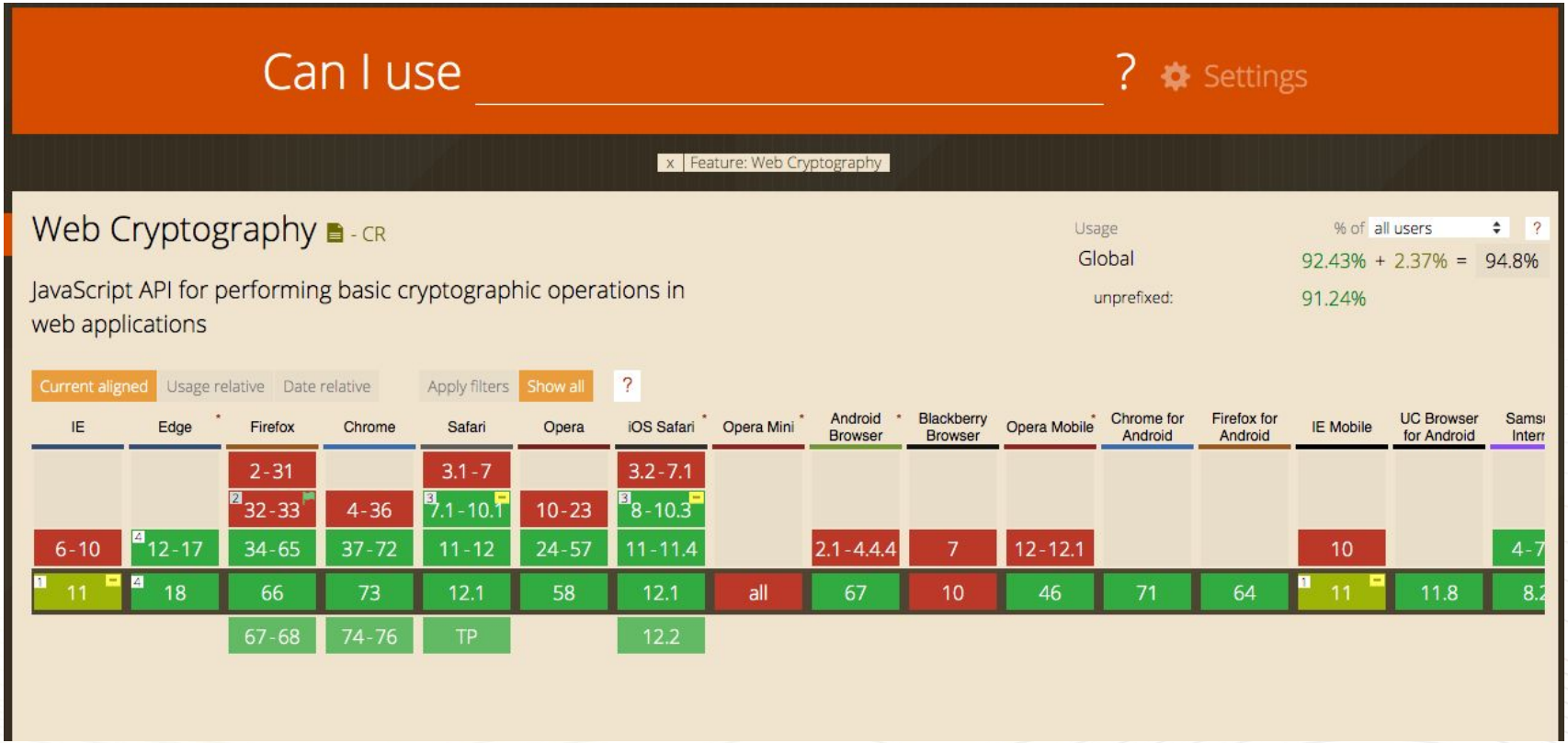
Web Crypto API

Web Crypto API

- Recommandation du W3C
  - https://www.w3.org/TR/WebCryptoAPI
- Version finale datant du 26 janvier 2017
  - commencée en décembre 2012
- Implémenté et supporté par la majorité des navigateurs

This specification describes a JavaScript API for performing basic cryptographic operations in web applications, such as hashing, signature generation and verification, and encryption and decryption. Additionally, it describes an API for applications to generate and/or manage the keying material necessary to perform these operations. Uses for this API range from user or service authentication, document or code signing, and the confidentiality and integrity of communications.

Web Crypto API



```
5  const webCrypto = window.crypto || window.msCrypto || window.webkitCrypto || window.mozCrypto;
```

Web Crypto API

```
ArrayBufferView getRandomValues(ArrayBufferView array);
```

Permet d'obtenir des valeurs pseudo-aléatoires cryptographiquement satisfaisantes.

```
SubtleCrypto subtle;

  Promise<any> encrypt(AlgorithmIdentifier algorithm,CryptoKey key, BufferSource data);

  Promise<any> decrypt(AlgorithmIdentifier algorithm,CryptoKey key,BufferSource data);

  Promise<any> sign(AlgorithmIdentifier algorithm,CryptoKey key,BufferSource data);

  Promise<any> verify(AlgorithmIdentifier algorithm, CryptoKey key,BufferSource signature,BufferSource data);

  Promise<any> generateKey(AlgorithmIdentifier algorithm,boolean extractable, sequence<KeyUsage> keyUsages );

  Promise<CryptoKey> importKey(KeyFormat format,(BufferSource or JsonWebKey) keyData, AlgorithmIdentifier
  algorithm,boolean extractable,sequence<KeyUsage> keyUsages );

  Promise<any> exportKey(KeyFormat format, CryptoKey key);

  ...
```

Web Crypto API

```
window.crypto.subtle
```

The SubtleCrypto interface provides a set of methods for dealing with low-level cryptographic primitives and algorithms. It is named `SubtleCrypto` to reflect the fact that many of these algorithms have subtle usage requirements in order to provide the required algorithmic security guarantees.

Algorithmes (peut-être) supportés

- dérivation de clé: PBKDF2, HKDF, DH, CONCAT, ECDH
- hachage: SHA-1, SHA-256, SHA-384, SHA-512
- key wrapping: AES-KW, AES-GCM, AES-CFB, AES-CBC, AES-CTR, RSA-OAEP
- sign / verify: HMAC, AES-CMAC, ECDSA, RSA-PSS, RSASSA-PKCS1-v1_5
- encrypt / decrypt: AES-GCM, AES-CBC, AES-CFB, AES-CTR, RSA-OAEP

Mais pourquoi c’est utile ?

Mais pourquoi c’est utile ?

- Globalement permet d’implémenter le chiffrement de bout en bout
  - chiffrement directement dans le client
  - utile pour implémenter des applications de type Zero knowledge architectures
- Par exemple
  - Multi-factor Authentication
  - Protected Document Exchange
  - Cloud Storage with encryption on the client
  - Document Signing
  - Data Integrity Protection
  - Secure Messaging

Notre application de messagerie e2ee

- chiffrée de bout en bout
  - tout se passe dans le client
- n’utilise pas de local storage ou autre pour stocker la session
  - tout est dérivé du mot de de passe utilisé pour se connecter
- volontairement simple (pour des raisons pédagogiques)
  - pas de chiffrement côté serveur pour permettre d’inspecter l’état
  - pas de signature des messages ni les hash de password
  - métadonnées explicites qu’il serait facile d’éviter
- nous ne sommes pas des experts en sécurité
- surement pas à l’état de l’art
  - nous n’implémentons pas le protocole Signal ;)

Notre application de messagerie e2ee

https://github.com/mathieuancelin/devoxx-web-crypto-demo

Démo

Lors de la création du compte

- Initialisation
  - hachage bcrypt du mot de passe
    - passwordHash = bcrypt.hash(password, bcrypt.generateSalt(10))
  - génération d'un sel propre à l'utilisateur
    - salt = bcrypt.generateSalt(10)
  - génération d'un couple de clés (publique/privée) propre à l'utilisateur
- Chiffrement de la clé privée et du sel
  - chiffrement AES du sel avec le mot de passe de l'utilisateur
    - encryptedSalt = aes.encrypt(salt, password)
  - création d'un hash bcrypt du mot de passe avec le sel pour chiffrer (AES) la clé privée avec
    - encryptedPrivateKey = aes.encrypt(privateKey, bcrypt.hash(password, salt))
- Stockage sur le serveur
  - email, nom, clé publique, passwordHash, encryptedSalt, encryptedPrivateKey

```
{
  "email": "bob@foo.bar",
  "name": "Bobby Boby",
  "password": "$2b$2$FtXnxAf6EU",
  "salt": "Y2JkZmZkMTg0NWF==",
  "publicKey": null,
  "privateKey": null
}
```

Lors de la création du compte

```
{
  "email": "bob@foo.bar",
  "name": "Bobby Boby",
  "password": "secret",
  "salt": null,
  "publicKey": null,
  "privateKey": null
}
```

Lors de la création du compte

- Initialisation
  - hachage bcrypt du mot de passe
    - passwordHash = bcrypt.hash(password, bcrypt.generateSalt(10))

```
{
  "email": "bob@foo.bar",
  "name": "Bobby Boby",
  "password": "$2b$2$FtXnxAf6EU",
  "salt": null,
  "publicKey": null,
  "privateKey": null
}
```

Lors de la création du compte

```
{
  "email": "bob@foo.bar",
  "name": "Bobby Boby",
  "password": "$2b$2$FtXnxAf6EU",
  "salt": "salty salt",
  "publicKey": null,
  "privateKey": null
}
```

- Initialisation
  - hachage bcrypt du mot de passe
    - passwordHash = bcrypt.hash(password, bcrypt.generateSalt(10))
  - génération d'un sel propre à l'utilisateur
    - salt = bcrypt.generateSalt(10)

Lors de la création du compte

```
{
  "email": "bob@foo.bar",
  "name": "Bobby Boby",
  "password": "$2b$2$FtXnxAf6EU",
  "salt": "salty salt",
  "publicKey": "a public key",
  "privateKey": "a private key"
}
```

- Initialisation
  - hachage bcrypt du mot de passe
    - passwordHash = bcrypt.hash(password, bcrypt.generateSalt(10))
  - génération d'un sel propre à l'utilisateur
    - salt = bcrypt.generateSalt(10)
  - génération d'un couple de clés (publique/privée) propre à l'utilisateur

Lors de la création du compte

```
{
  "email": "bob@foo.bar",
  "name": "Bobby Boby",
  "password": "$2b$2$FtXnxAf6EU",
  "salt": "wMTRhODAwYWVlYTA3Zj",
  "publicKey": "a public key",
  "privateKey": "a private key"
}
```

- Initialisation
  - hachage bcrypt du mot de passe
    - passwordHash = bcrypt.hash(password, bcrypt.generateSalt(10))
  - génération d'un sel propre à l'utilisateur
    - salt = bcrypt.generateSalt(10)
  - génération d'un couple de clés (publique/privée) propre à l'utilisateur
- Chiffrement de la clé privée et du sel
  - chiffrement AES du sel avec le mot de passe de l'utilisateur
    - encryptedSalt = aes.encrypt(salt, password)

Lors de la création du compte

```
{
  "email": "bob@foo.bar",
  "name": "Bobby Boby",
  "password": "$2b$2$FtXnxAf6EU",
  "salt": "wMTRhODAwYWVlYTA3Zj",
  "publicKey": "a public key",
  "privateKey": "BueD6aAB2L6"
}
```

- Initialisation
  - hachage bcrypt du mot de passe
    - passwordHash = bcrypt.hash(password, bcrypt.generateSalt(10))
  - génération d'un sel propre à l'utilisateur
    - salt = bcrypt.generateSalt(10)
  - génération d'un couple de clés (publique/privée) propre à l'utilisateur
- Chiffrement de la clé privée et du sel
  - chiffrement AES du sel avec le mot de passe de l'utilisateur
    - encryptedSalt = aes.encrypt(salt, password)
  - création d'un hash bcrypt du mot de passe avec le sel pour chiffrer (AES) la clé privée avec
    - encryptedPrivateKey = aes.encrypt(privateKey, bcrypt.hash(password, salt))

Lors de la création du compte

```
{
  "email": "bob@foo.bar",
  "name": "Bobby Boby",
  "password": "$2b$2$FtXnxAf6EU",
  "salt": "wMTRhODAwYWVlYTA3Zj",
  "publicKey": "a public key",
  "privateKey": "BueD6aAB2L6"
}
```

- Initialisation
  - hachage bcrypt du mot de passe
    - passwordHash = bcrypt.hash(password, bcrypt.generateSalt(10))
  - génération d'un sel propre à l'utilisateur
    - salt = bcrypt.generateSalt(10)
  - génération d'un couple de clés (publique/privée) propre à l'utilisateur
- Chiffrement de la clé privée et du sel
  - chiffrement AES du sel avec le mot de passe de l'utilisateur
    - encryptedSalt = aes.encrypt(salt, password)
  - création d'un hash bcrypt du mot de passe avec le sel pour chiffrer (AES) la clé privée avec
    - encryptedPrivateKey = aes.encrypt(privateKey, bcrypt.hash(password, salt))
- Stockage sur le serveur
  - email, nom, clé publique, passwordHash, encryptedSalt, encryptedPrivateKey

```
7  const enc = new TextEncoder();
8  const dec = new TextDecoder("utf-8");
9
10 function stringToBytes(ascii) {
11   return enc.encode(ascii);
12 }
13
14 function bytesToBase64String(bytes) {
15   return window.btoa(new Uint8Array(bytes).reduce((str, byte) => str + byte.toString(16).padStart(2, '0'), '')).trim();
16 }
17
18 function base64StringToBytes(hex) {
19   return new Uint8Array(window.atob(hex).match(/.{1,2}/g).map(byte => parseInt(byte, 16)));
20 }
21
22 function bytesToString(bytes) {
23   return dec.decode(bytes);
24 }
```

```
92    genKeyPair() {
93      return webCrypto.subtle.generateKey({
94          name: this.rsaName,
95          modulusLength: 2048, //can be 1024, 2048, or 4096
96          publicExponent: new Uint8Array([0x01, 0x00, 0x01]),
97          hash: { name: "SHA-256" }, //can be "SHA-1", "SHA-256", "SHA-384", or "SHA-512"
98        },
99        true, //whether the key is extractable (i.e. can be used in exportKey)
100       ["encrypt", "decrypt"] //must be ["encrypt", "decrypt"] or ["wrapKey", "unwrapKey"]
101     )
102     .then((key) => {
103       return this.exportAsJwk(key.publicKey).then(publicKey => {
104         return this.exportAsJwk(key.privateKey).then(privateKey => {
105           return {
106             publicKey,
107             privateKey
108           }
109         });
110       });
111     })
112     .catch((err) => {
113       console.error(`[RSA] Error while gen key pair ${err.message}`, err);
114     });
115   }
```

```
66    exportAsJwk(key) {
67      return webCrypto.subtle.exportKey(
68        "jwk", //can be "jwk" (public or private), "spki" (public only), or "pkcs8" (private only)
69        key //can be a publicKey or privateKey, as long as extractable was true
70      )
71      .catch((err) => {
72        console.error(`[RSA] Error while exporting key ${err.message}`, err);
73      });
74    }
```

Lors du login

- Vérification serveur
  - couple email / bcrypt.compareSync(password, passwordHash)
  - retourne (email, nom, clé publique, passwordHash, encryptedSalt, encryptedPrivateKey)
- Déchiffrement du sel
  - salt = aes.decrypt(user.encryptedSalt, password)
- Déchiffrement de la clé privée
  - privateKey = aes.decrypt(user.encryptedPrivateKey, bcrypt.hash(password, salt))
- On oublie le mot de passe !!!

Envoi d’un message

- Génération d’une clé aléatoire
  - randomKey = crypto.getRandomValues()
- Chiffrement AES du contenu du message avec la clé aléatoire
  - encryptedMessage = aes.encrypt(message, randomKey)
- Chiffrement RSA de la clé aléatoire avec la clé publique du destinataire
  - encryptedRandomKey = rsa.encrypt(randomKey, alicePublicKey)
- Stockage serveur
  - (from, to, date, id, encryptedRandomKey, encryptedMessage)

Envoi d’un message

```
{
  "key": null,
  "content": "un message",
  "from": "bob@foo.bar",
  "to": "bob@foo.bar",
  "at": 1555451160579,
  "id": "Fv+mPQxNtqJpa9E28wrW2w=="
}
```

Envoi d’un message

- Génération d’une clé aléatoire (unique / message)
  - randomKey = crypto.getRandomValues()

```
{
  "key": "a random value",
  "content": "un message",
  "from": "bob@foo.bar",
  "to": "bob@foo.bar",
  "at": 1555451160579,
  "id": "Fv+mPQxNtqJpa9E28wrW2w=="
}
```

Envoi d’un message

- Génération d’une clé aléatoire
  - randomKey = crypto.getRandomValues()
- Chiffrement AES du contenu du message avec la clé aléatoire
  - encryptedMessage = aes.encrypt(message, randomKey)


```
{
  "key": "a random value",
  "content": "ZWVhOTU4ODUxZDFi",
  "from": "bob@foo.bar",
  "to": "bob@foo.bar",
  "at": 1555451160579,
  "id": "Fv+mPQxNtqJpa9E28wrW2w=="
}
```

Envoi d’un message


```
{
  "key": "MGQ1MTUwZTU2ZWRmNWMxY",
  "content": "ZWVhOTU4ODUxZDFi",
  "from": "bob@foo.bar",
  "to": "bob@foo.bar",
  "at": 1555451160579,
  "id": "Fv+mPQxNtqJpa9E28wrW2w=="
}
```


- Génération d’une clé aléatoire
  - randomKey = crypto.getRandomValues()
- Chiffrement AES du contenu du message avec la clé aléatoire
  - encryptedMessage = aes.encrypt(message, randomKey)
- Chiffrement RSA de la clé aléatoire avec la clé publique du destinataire
  - encryptedRandomKey = rsa.encrypt(randomKey, alicePublicKey)

Envoi d’un message


```
{
  "key": "MGQ1MTUwZTU2ZWRmNWMxY",
  "content": "ZWVhOTU4ODUxZDFi",
  "from": "bob@foo.bar",
  "to": "bob@foo.bar",
  "at": 1555451160579,
  "id": "Fv+mPQxNtqJpa9E28wrW2w=="
}
```


- Génération d’une clé aléatoire
  - randomKey = crypto.getRandomValues()
- Chiffrement AES du contenu du message avec la clé aléatoire
  - encryptedMessage = aes.encrypt(message, randomKey)
- Chiffrement RSA de la clé aléatoire avec la clé publique du destinataire
  - encryptedRandomKey = rsa.encrypt(randomKey, alicePublicKey)
- Stockage serveur
  - (from, to, date, id, encryptedRandomKey, encryptedMessage)

```
172    sendMessage(to, content) {
173      return this.server.getPublicKey(to).then(res => {
174        const toPublicKey = res.publicKey;
175        if (toPublicKey) {
176          return this.encryptMessage(content, toPublicKey).then(encryptedMessage => {
177            encryptedMessage.from = this.email;
178            encryptedMessage.to = to;
179            encryptedMessage.at = Date.now();
180            encryptedMessage.id = generateRandomKey();
181            if (to === this.email) {
182              return this.server.sendMessage(this.email, encryptedMessage);
183            } else {
184              return this.encryptMessage(content).then(selfEncryptedMessage => {
185                selfEncryptedMessage.from = this.email;
186                selfEncryptedMessage.to = to;
187                selfEncryptedMessage.at = encryptedMessage.at;
188                selfEncryptedMessage.id = encryptedMessage.id;
189                return this.server.sendMessage(this.email, encryptedMessage, selfEncryptedMessage);
190              });
191            }
192          });
193        } else {
194          console.log('No public key for user', to);
195          return null;
196        }
197      });
198    }
199  }
```

```
172   sendMessage(to, content) {
173     return this.server.getPublicKey(to).then(res => {
174       const toPublicKey = res.publicKey;
175       if (toPublicKey) {
176         return this.encryptMessage(content, toPublicKey).then(encryptedMessage => {
177           encryptedMessage.from = this.email;
178           encryptedMessage.to = to;
179           encryptedMessage.at = Date.now();
180           encryptedMessage.id = generateRandomKey();
181           if (to === this.email) {
182             return this.server.sendMessage(this.email, encryptedMessage);
183           } else {
184             return this.encryptMessage(content).then(selfEncryptedMessage => {
185               selfEncryptedMessage.from = this.email;
186               selfEncryptedMessage.to = to;
187               selfEncryptedMessage.at = encryptedMessage.at;
188               selfEncryptedMessage.id = encryptedMessage.id;
189               return this.server.sendMessage(this.email, encryptedMessage, selfEncryptedMessage);
190             });
191           }
192         });
193       } else {
194         console.log('No public key for user', to);
195         return null;
196       }
197     });
198   }
199 }
```

RSA Encrypt (RSA-OAEP)

```
30    encrypt(text, publicKey) {
31      return this.importFromJwk(publicKey, true).then(key => {
32        return webCrypto.subtle.encrypt(
33          {
34              name: this.rsaName,
35          },
36          key, //from generateKey or importKey above
37          stringToBytes(text),
38        )
39        .then((encrypted) => {
40          return bytesToBase64String(encrypted);
41        })
42        .catch((err) => {
43          console.error(`[RSA] Error while decrypt ${err.message}`, err);
44        });
45      });
46    }
```

RSA Encrypt (RSA-OAEP)

```
76    importFromJwk(key, pub) {
77      return webCrypto.subtle.importKey(
78        "jwk", //can be "jwk" (public or private), "spki" (public only), or "pkcs8" (private only)
79        key,
80        {   //these are the algorithm options
81          name: this.rsaName,
82          hash: {name: "SHA-256"}, //can be "SHA-1", "SHA-256", "SHA-384", or "SHA-512"
83        },
84        true, //whether the key is extractable (i.e. can be used in exportKey)
85        [pub ? "encrypt" : "decrypt"]
86      )
87      .catch((err) => {
88        console.error(`[RSA] Error while importing key ${err.message}`, err);
89      });
90    }
```

AES Encrypt (AES-CTR)

```
142   encrypt(text, masterkey) {
143     return this.deriveKey(masterkey).then(key => {
144       return webCrypto.subtle.encrypt(
145         {
146           name: this.aesName,
147           //Don't re-use counters!
148           //Always use a new counter every time your encrypt!
149           counter: new Uint8Array(16),
150           length: 128, //can be 1-128
151         },
152         key, //from generateKey or importKey above
153         stringToBytes(text) //ArrayBuffer of data you want to encrypt
154       )
155       .then((encrypted) => {
156         return bytesToBase64String(encrypted);
157       })
158       .catch(function(err){
159         console.error(`[AES] error while encrypt ${err.message}`, err);
160       });
161     });
162   }
```

Lecture d’un message

- Déchiffrement RSA de la clé aléatoire du message avec la clé privée
  - randomKey = rsa.decrypt(message.encryptedRandomKey, alicePrivateKey)
- Déchiffrement AES du contenu du message avec la clé aléatoire
  - content = aes.decrypt(message.encryptedMessage, randomKey)

Lecture d’un message


```
{
  "key": "MGQ1MTUwZTU2ZWRmNWMxY",
  "content": "ZWVhOTU4ODUxZDFi",
  "from": "bob@foo.bar",
  "to": "bob@foo.bar",
  "at": 1555451160579,
  "id": "Fv+mPQxNtqJpa9E28wrW2w=="
}
```

Lecture d’un message

- Déchiffrement RSA de la clé aléatoire du message avec la clé privée
  - randomKey = rsa.decrypt(message.encryptedRandomKey, alicePrivateKey)

```
{
  "key": "a random value",
  "content": "ZWVhOTU4ODUxZDFi",
  "from": "bob@foo.bar",
  "to": "bob@foo.bar",
  "at": 1555451160579,
  "id": "Fv+mPQxNtqJpa9E28wrW2w=="
}
```

Lecture d’un message

- Déchiffrement RSA de la clé aléatoire du message avec la clé privée
  - randomKey = rsa.decrypt(message.encryptedRandomKey, alicePrivateKey)
- Déchiffrement AES du contenu du message avec la clé aléatoire
  - content = aes.decrypt(message.encryptedMessage, randomKey)

```
{
  "key": "a random value",
  "content": "un message",
  "from": "bob@foo.bar",
  "to": "bob@foo.bar",
  "at": 1555451160579,
  "id": "Fv+mPQxNtqJpa9E28wrW2w=="
}
```

RSA Decrypt (RSA-OAEP)

```
48    decrypt(encdata, privateKey) {
49      return this.importFromJwk(privateKey, false).then(key => {
50        return webCrypto.subtle.decrypt(
51          {
52            name: this.rsaName,
53          },
54          key, //from generateKey or importKey above
55          base64StringToBytes(encdata) //ArrayBuffer of the data
56        )
57        .then((decrypted) => {
58          return bytesToString(decrypted);
59        })
60        .catch((err) => {
61          console.error(`[RSA] Error while decrypt ${err.message}`, err);
62        });
63      });
64    }
```

AES Decrypt (AES-CTR)

```
164   decrypt(encdata, masterkey) {
165     return this.deriveKey(masterkey).then(key => {
166       return webCrypto.subtle.decrypt(
167         {
168             name: this.aesName,
169             counter: new Uint8Array(16), //The same counter you used to encrypt
170             length: 128, //The same length you used to encrypt
171         },
172         key, //from generateKey or importKey above
173         base64StringToBytes(encdata) //ArrayBuffer of the data
174       )
175       .then((decrypted) => {
176         return bytesToString(decrypted);
177       })
178       .catch((err) => {
179         console.error(`[AES] error while decrypt ${err.message}`, err);
180       });
181     });
182   }
```

Utilisation de la Web Crypto API

https://github.com/diafygi/webcrypto-example

Et du coup, on se lance ?

Et du coup, on se lance ?

oui et non ...

Utilisez des librairies

Il est très facile de se tromper en utilisant la Web Crypto API (comme vous l'avez vu, il faut être un peu du coin)

De nombreuses librairies existent pour éviter d'avoir à gérer tout ça à la main. Parfois ce sont des wrapper de la Web Crypto API, parfois des ré-implémentations complètes

- JSEncrypt
- js-nacl
- openpgp.js
- ...

https://gist.github.com/jo/8619441

exemple: JSEncrypt

```
116  const encrypt = new JSEncrypt();
117  encrypt.setPublicKey(pubkey);
118  const ciphertext = encrypt.encrypt(message);
119
120  const decrypt = new JSEncrypt();
121  decrypt.setPrivateKey(privkey);
122  const plaintext = decrypt.decrypt(ciphertext);
123
124  console.log(plaintext == message ? 'It Works!' : 'Error with decryption');
```

```
import * as openpgp from 'openpgp';
openpgp.initWorker({ path: 'openpgp/openpgp.worker.js' });

openpgp.generateKey({
  numBits: 2048,
  userIds: [{ name: 'bob', email: 'bob@foo.bar' }]
}).then(pair => {
  return openpgp.key.readArmored(pair.publicKeyArmored).then(keys => {
    return openpgp.encrypt({
      message: openpgp.message.fromText('Hello World!'),
      publicKeys: keys.keys,
    }).then(encrypted => {
      return openpgp.stream.readToEnd(encrypted.data);
    })
  }).then(encryptedText => {
    console.log('encryptedText', encryptedText);
    return openpgp.key.readArmored(pair.privateKeyArmored).then(keys => {
      return openpgp.message.readArmored(encdata).then(m => {
        return openpgp.decrypt({
          message: m,
          privateKeys: keys.keys,
        }).then(decrypted => {
          return openpgp.stream.readToEnd(decrypted.data);
        }).then(decryptedText => {
          console.log('decryptedText', decryptedText);
        });
      });
    });
  });
});
```

Démo openpgp.js

Questions ?