COMET

Competence Centers for Excellent Technologies

End-to-end File Encryption in the Web Browser – A Case Study

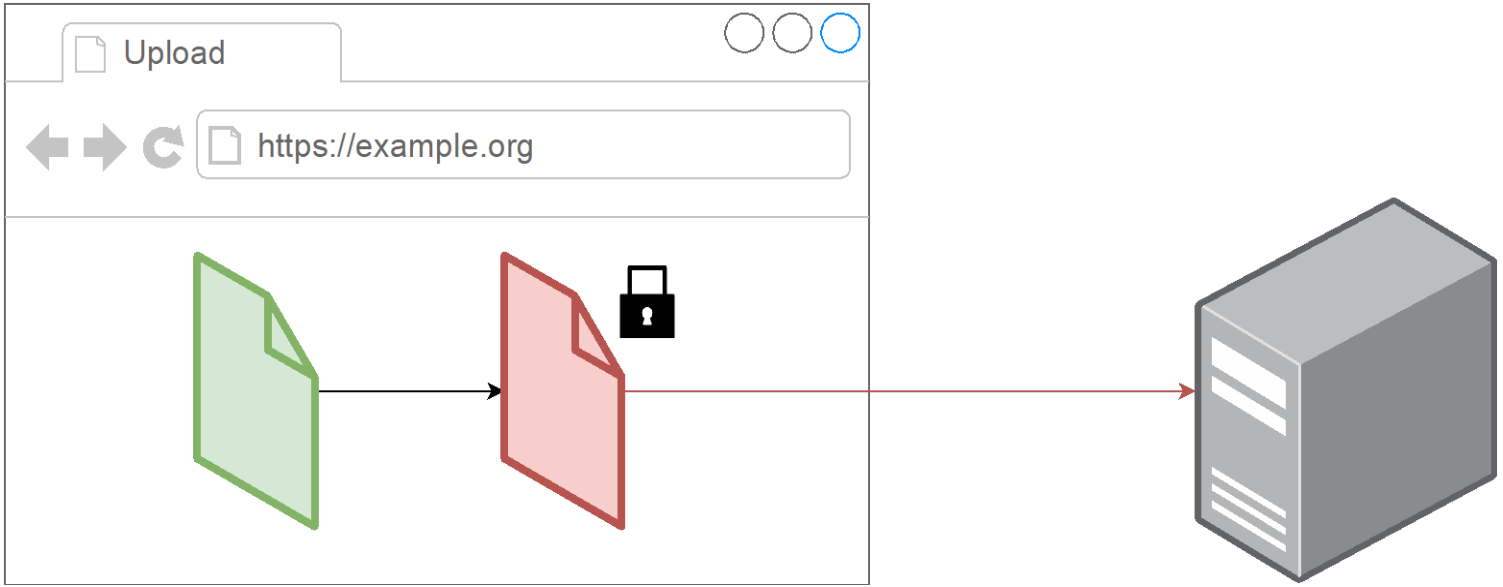
Thomas Konrad, SBA Research

Vue.js Meetup Vienna, Feb 11, 2020

Bundesministerium Verkehr, Innovation und Technologie
Bundesministerium Digitalisierung und Wirtschaftsstandort
FFG Forschung wirkt.
wirtschaftsagentur wien Ein Fonds der Stadt Wien
FWF Der Wissenschaftsfonds.
netidee OPEN INNOVATIONS

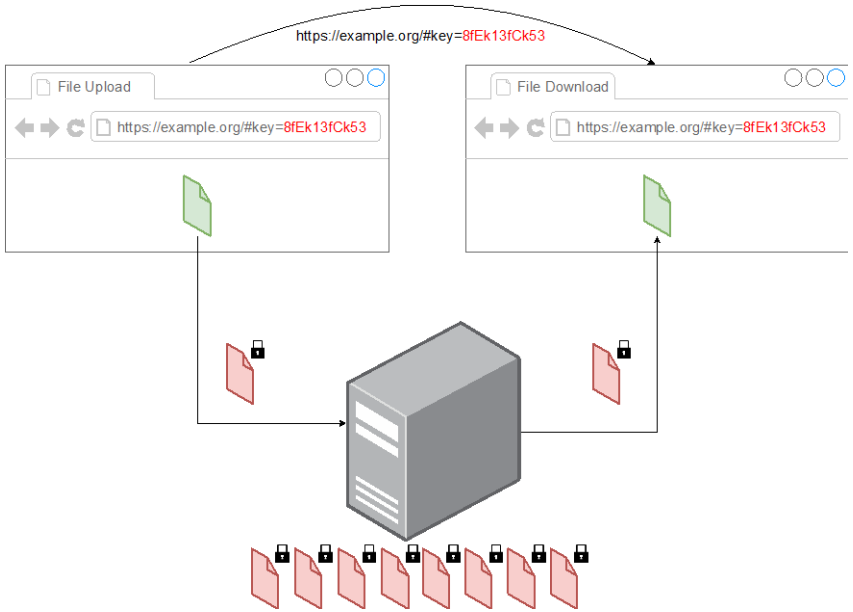
~$ whoami
Thomas Konrad
~$ id
uid=123
gid=1(Vienna, Austria)
gid=2(SBA Research)
gid=3(Software Security)
gid=4(Penetration Testing)
gid=5(Secure Development Lifecycle)
gid=6(Security Training)
gid=7(sec4dev Conference & Bootcamp)

What we'd like to do

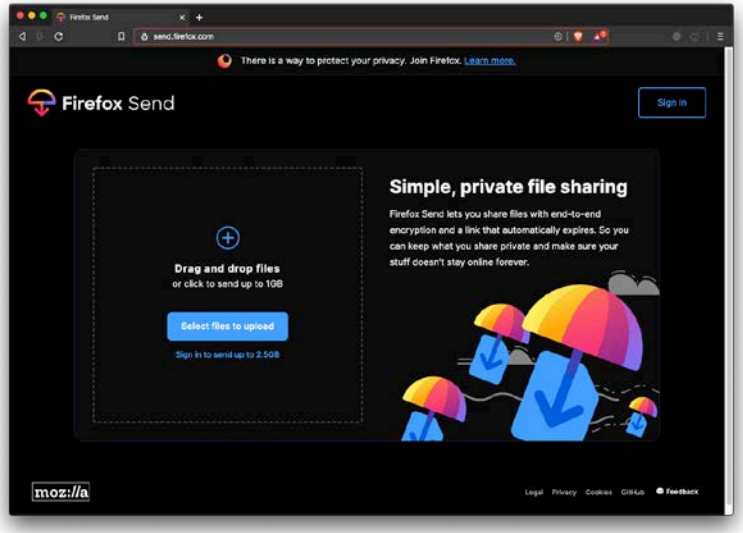


Why all the Fuzz?

- Beecozz Sucurity!
- **No, seriously**
  - To lower the attack surface significantly!
  - File read access on the server gets worthless to attackers
  - Maybe better deployment options



I admit: I didn’t invent this.



- **Firefox Send**
- End-to-end encrypted file sharing
- Basic concepts in this talk are taken from there
- https://send.firefox.com
- https://github.com/mozilla/send
- **Good stuff!**

Requirements

- Ensure confidentiality, integrity, and authenticity
- Encrypt the file in the browser
- Support big files (> 10 GB)
- Support old browsers as well as possible
- Securely distribute keys

Ensure Confidentiality, Integrity, and Authenticity

First things first: **Randomness**.

1110 1101 0001 1101

0000 0000 0000 0000

"There is no such thing as a random number – there are only methods to produce random numbers."

– John von Neumann (1961)

Generate Secure Random Bytes in the Browser

```
public static generateRandomBytes(
  length: number
): Uint8Array {
  return crypto.getRandomValues(
    new Uint8Array(length)
  );
}
```

GitHub Gist - Random Number Generator in TypeScript:
https://gist.github.com/thomaskonrad/cfe4c9f6fd26f4382df1f8de3a2b97e8

Ensure Confidentiality, Integrity, and Authenticity

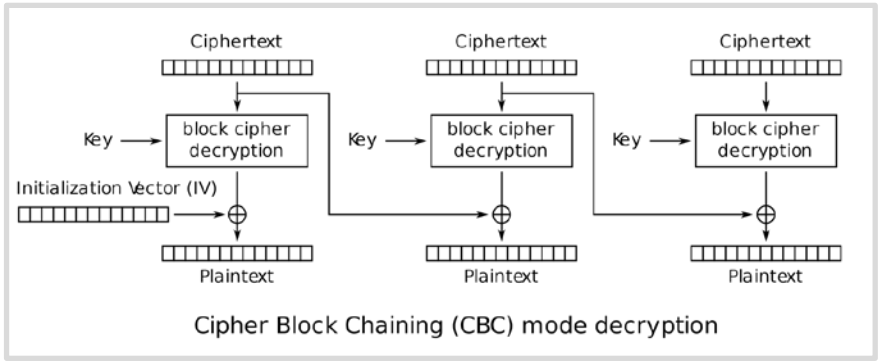
Let's take as an example **AES** in **CBC** mode.

What do you think we can guarantee with it?

✓ **Confidentiality**

✗ **Integrity**

✗ **Authenticity**



Cipher Block Chaining (CBC) mode decryption

CBC (and others) Do Not Guarantee Integrity

- The ciphertext can be manipulated, and decryption will not fail.
- **Solution**
  - Authenticated Encryption with Associated Data (AEAD)
  - Examples: AES-GCM, ChaCha20-Poly1305

Requirements

- Ensure confidentiality, integrity, and authenticity
- Encrypt the file in the browser
- Support big files (> 10 GB)
- Support old browsers as well as possible
- Securely distribute keys

Symmetric AEAD Encryption in the Browser 1/2

```
export default class AEAD {
  public static readonly KEY_LENGTH_IN_BYTES = 16;
  public static readonly IV_LENGTH_IN_BYTES = 16;
  public static readonly TAG_LENGTH_IN_BYTES = 16;
  private static readonly ALGORITHM = 'AES-GCM';
  private readonly secretKey: CryptoKey;
  private readonly tagLengthInBytes: number;

  public constructor(secretKey: CryptoKey, tagLengthInBytes = AEAD.TAG_LENGTH_IN_BYTES) {
    this.secretKey = secretKey;
    this.tagLengthInBytes = tagLengthInBytes;
  }

  public static async getCryptoKeyFromRawKey(rawKey: Uint8Array): Promise<CryptoKey> {
    return await crypto.subtle.importKey(
      'raw',
      rawKey,
      {
        name: this.ALGORITHM,
      },
      true,
      ['encrypt', 'decrypt'],
    );
  }
```

GitHub Gist - Authenticated Secret Key Cryptography (AEAD) in TypeScript: https://gist.github.com/thomaskonrad/c8ed4d73bb200ecc9ecdf5b85e2eb7f0

Symmetric AEAD Encryption in the Browser 2/2

```
public async encrypt(iv: Uint8Array, data: Uint8Array): Promise<ArrayBuffer> {
  return await crypto.subtle.encrypt({
    name: AuthenticatedSecretKeyCryptography.ALGORITHM,
    iv,
    tagLength: this.tagLengthInBytes * 8,
   },
   this.secretKey,
   data,
  );
 }

 public async decrypt(iv: Uint8Array, data: Uint8Array): Promise<ArrayBuffer> {
  return await crypto.subtle.decrypt({
    name: AuthenticatedSecretKeyCryptography.ALGORITHM,
    iv,
    tagLength: this.tagLengthInBytes * 8,
   },
   this.secretKey,
   data,
  );
 }
}
```

GitHub Gist - Authenticated Secret Key Cryptography (AEAD) in TypeScript: https://gist.github.com/thomaskonrad/c8ed4d73bb200ecc9ecdf5b85e2eb7f0

Requirements

- Ensure confidentiality, integrity, and authenticity ✓
- Encrypt the file in the browser ✓
- Support big files (> 10 GB)
- Support old browsers as well as possible
- Securely distribute keys

Upload: Chunk and Encrypt

Upload: Splitting a File using the FileReader API

```
<template>
  <input type="file" @change="files = $event.target.files"/>
</template>

// ...

const readChunk = () => {
  const r = new FileReader();
  const blob = file.slice(offset, chunkSize + offset);
  r.onload = readEventHandler;
  r.readAsArrayBuffer(blob);
};
```

GitHub Gist - Split files into chunks of a given size using the FileReader API: https://gist.github.com/thomaskonrad/8ced18322ebfcba47d7e8765176eeefb

Upload: Splitting a File using the FileReader API

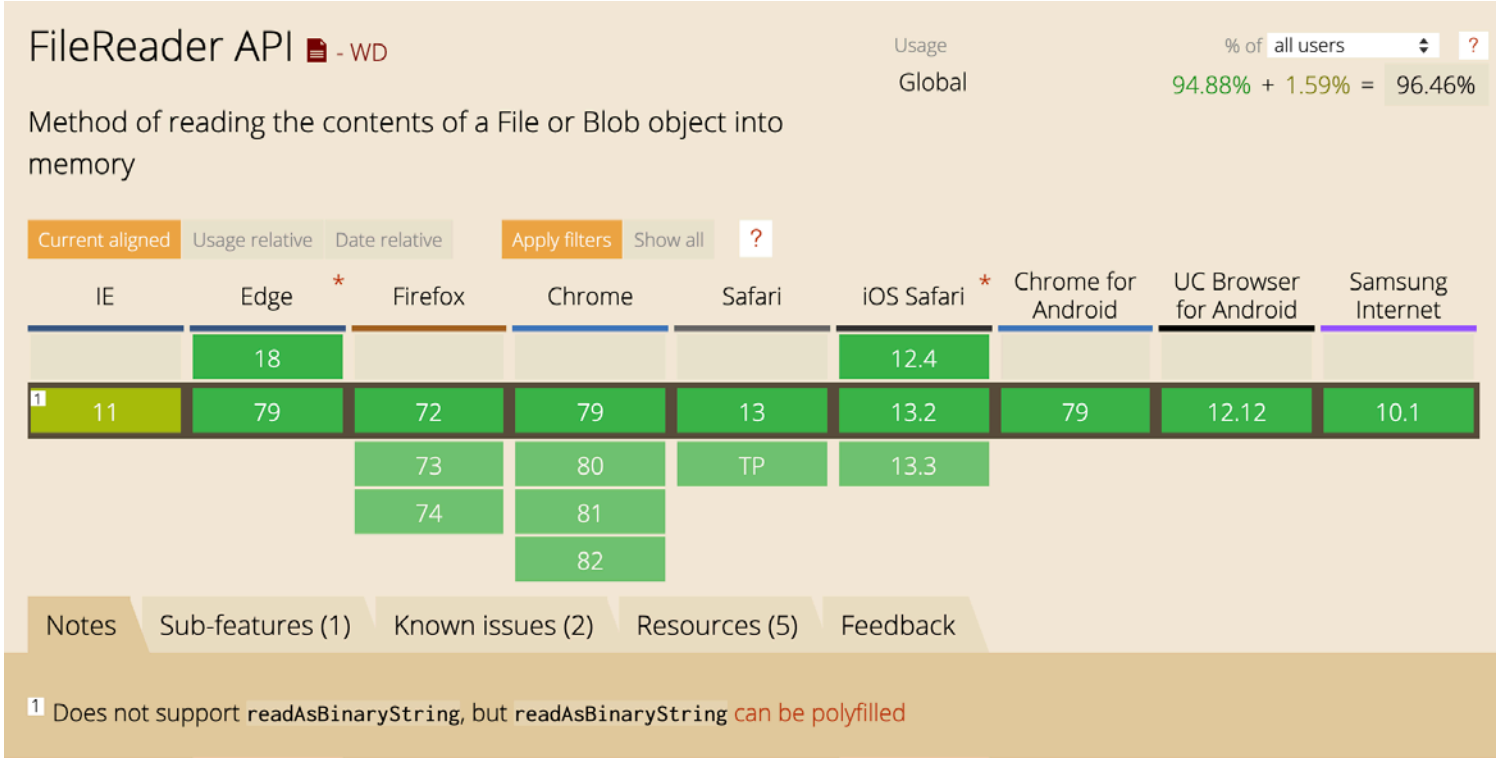
```
const readEventHandler = async (evt: any) => {
  if (evt.target.error == null) {
    const sequenceNumber = (offset / chunkSize);
    offset += evt.target.result.byteLength;
    let data = new Uint8Array(evt.target.result);

    data = await encrypt(data, sequenceNumber);
    await upload(data, sequenceNumber);
  }
  // Error handling, termination, read next chunk.
};
```

GitHub Gist - Split files into chunks of a given size using the FileReader API: https://gist.github.com/thomaskonrad/8ced18322ebfcba47d7e8765176eeefb

Upload: Creating the Encryption Transformer

```
const encrypt =
  async (chunk: Uint8Array, chunkIndex: number) => {
    return await this.keyChain.encrypt(
      chunk,
      file.nonce, // A random, 12-byte nonce!
      'fileContents',
      chunkIndex  // This will be part of the nonce.
    );
  };
```

FileReader API: Browser Support



Download: Download as Blob?

```
async blobDownload(file: File) {
  const response = await this.$http.get(
    `/api/v1/files/${file.id}`,
    { responseType: 'blob' },
  );

  const cleartext = await this.keyChain.getDecryptedResponseData(
    response.data,
    file.nonce,
    file.totalClearTextSizeInBytes,
  );

  await saveFile(cleartext, file.name, file.mimeType);
}
```

Download: Creating a ”Save as” Dialog

```
export default async function saveFile(plaintext: ArrayBuffer, fileName: string, fileType:
string) {
  return new Promise((resolve, reject) => {
    const dataView = new DataView(plaintext);
    const blob = new Blob([dataView], { type: fileType });

    const downloadUrl = URL.createObjectURL(blob);
    const a = document.createElement('a');
    a.href = downloadUrl;
    a.download = fileName;
    document.body.appendChild(a);
    a.click();
    URL.revokeObjectURL(downloadUrl);
    setTimeout(resolve, 100);
  });
}
```

GitHub Gist: Downloading an Array Buffer via a "Save as" Dialog in the Browser. https://gist.github.com/thomaskonrad/3777256a86d9f5b0472b3c2440cffee

Download: Downsides of this Approach

- A 10 GB file will use 20 GB+ of memory
- Using this approach, we cannot stream the data
- Enter **Service Workers**!

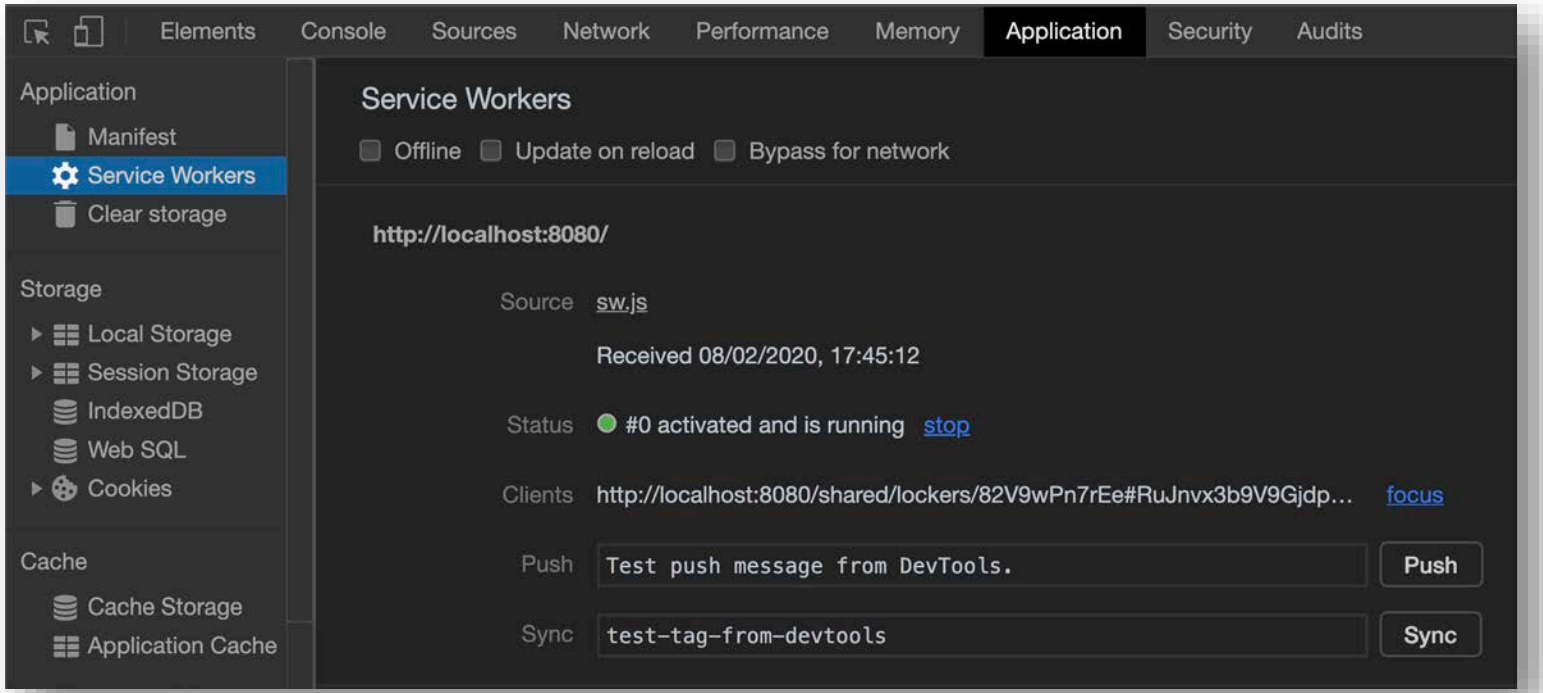
Service Workers: A Quick Primer

- A **piece of JavaScript** code that runs in the background of your website
- It acts as a **proxy server** between your site and the server
- It runs in a worker context and has **no DOM access**

Service Workers: A Quick Primer

- **Service Worker use cases**
  - Intercepting and modifying requests
  - Caching resources
  - Convert images
  - Can make your app work even if no network is available (Progressive Web Apps, PWA)

Service Workers: A Quick Primer



Registering a Service Worker

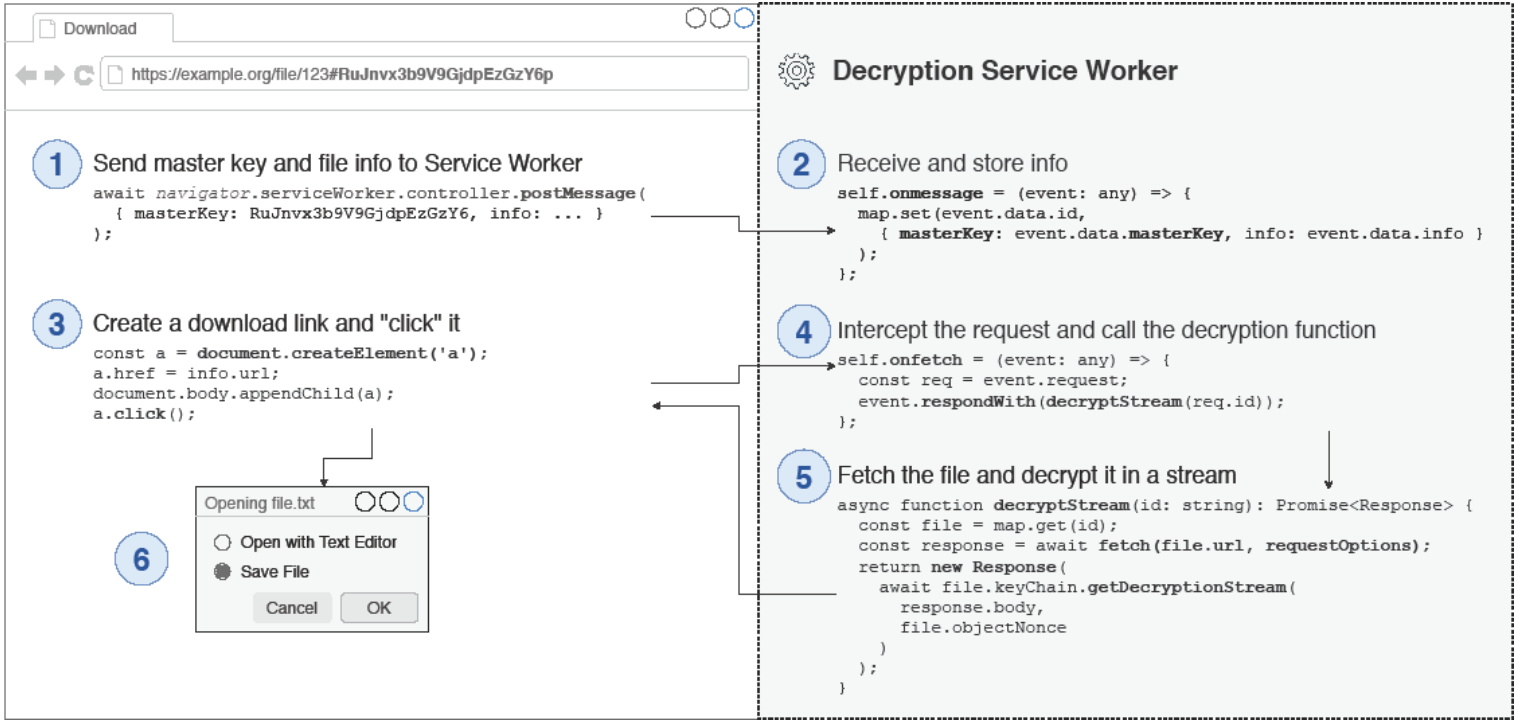
```
if ('serviceWorker' in navigator) {
  // Register a service worker hosted at the root of the
  // site using a more restrictive scope.
  navigator.serviceWorker.register('/sw.js', {
    scope: './'
  }).then((registration) => {
    console.log('Registration succeeded:', registration);
  }, /*catch*/ (error) => {
    console.log('Service worker registration failed:', error);
  });
} else {
  console.log('Service workers are not supported.');
}
```

Source: https://developer.mozilla.org/en-US/docs/Web/API/ServiceWorkerContainer/register

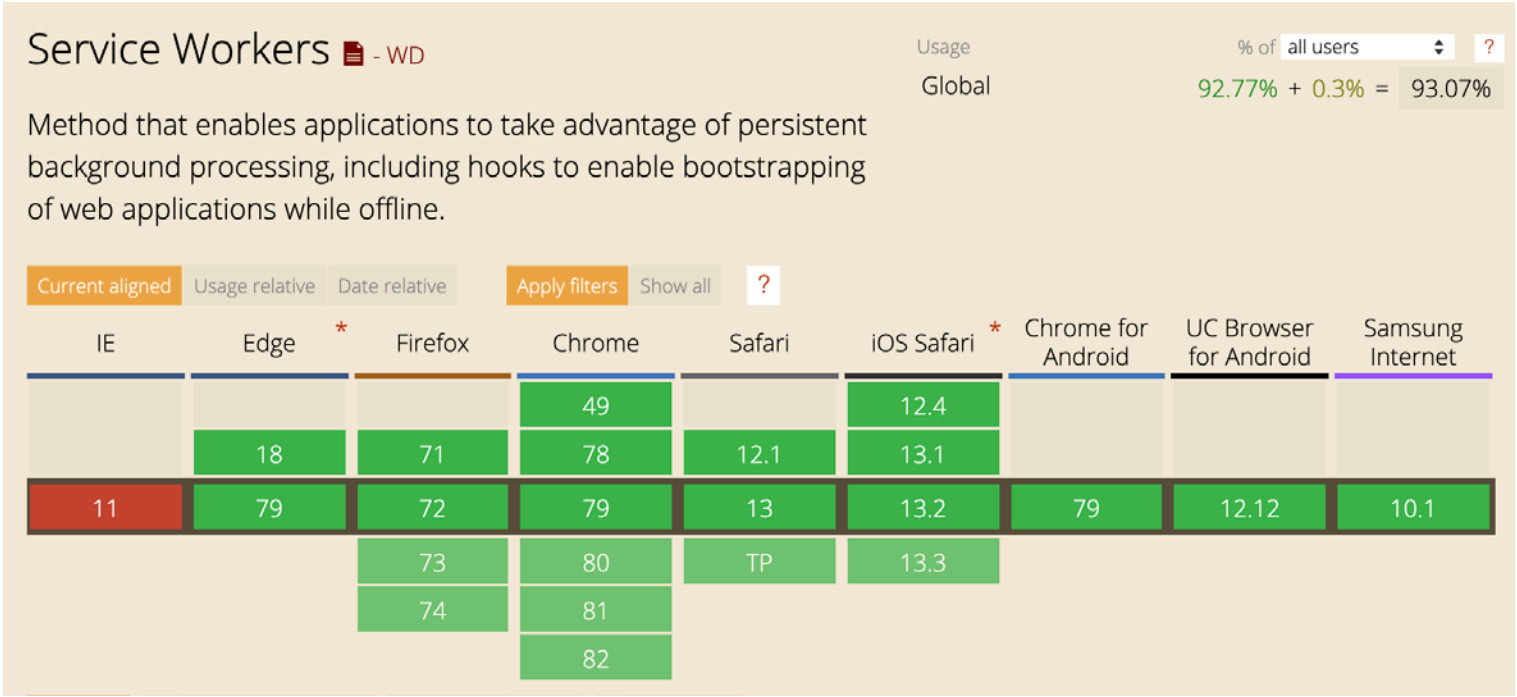
Activate the Service Worker Immediately

```
// In sw.js

self.addEventListener('install', (event) => {
  // Activate the Service Worker as soon as it's
  // finished installing.
  event.waitUntil(self.skipWaiting());
});

self.addEventListener('activate', (event) => {
  // Make the Service Worker become immediately
  // available to all pages.
  event.waitUntil(self.clients.claim());
});
```

Download: Decrypt the File in the SW



Service Workers: Browser Support



Download: The `ReadableStream` API

- **Readable stream of byte data**
- The Fetch API returns a `ReadableStream` object through the `body` property of the `Response` object

```
const response = await fetch(file.url, requestOptions);
const readableStream: ReadableStream<Uint8Array> = response.body;
```

Download: The Decryption Stream

```
public async getDecryptionStream(
  responseBody: ReadableStream<Uint8Array>,
  objectNonce: string,
): Promise<ReadableStream> {
  await this.initializeAEADIfNotInitialized();

  const slicesStream = transformStream(
    responseBody,
    new StreamSlicer(this.chunkSizeInBytes + this.tagLengthInBytes),
  );

  return transformStream(
    slicesStream,
    new DecryptStreamController(this, objectNonce, 'fileContents'),
  );
}
```

GitHub Gist - Splits a `ReadableStream` into chunks of a given size.:
https://gist.github.com/thomaskonrad/b8f30e3f18ea2f538bdf422203bdc473

Download: The `DecryptStreamController`

```
export default class DecryptStreamController {
  public async transform(chunk: any, controller: any){
    const clearText = await this.keyChain.decrypt(
      chunk,
      this.objectNonce,
      this.attributeName,
      this.chunkIndex
    );
    controller.enqueue(Buffer.from(clearText));
    this.chunkIndex ++;
  }
}
```

`ReadableStream` API: Browser Support

Requirements

- Ensure confidentiality, integrity, and authenticity ✓
- Encrypt the file in the browser ✓
- Support big files (> 10 GB) ✓
- Support old browsers as well as possible
- Securely distribute keys

Supporting Old Browsers

- **Upload**
  - The FileReader API is well supported
  - So no problem!
- **Download**
  - Requires Service Workers and *native* streams
  - Just Chromium-based and Gecko-based browsers
  - **But we can use the blob download!**

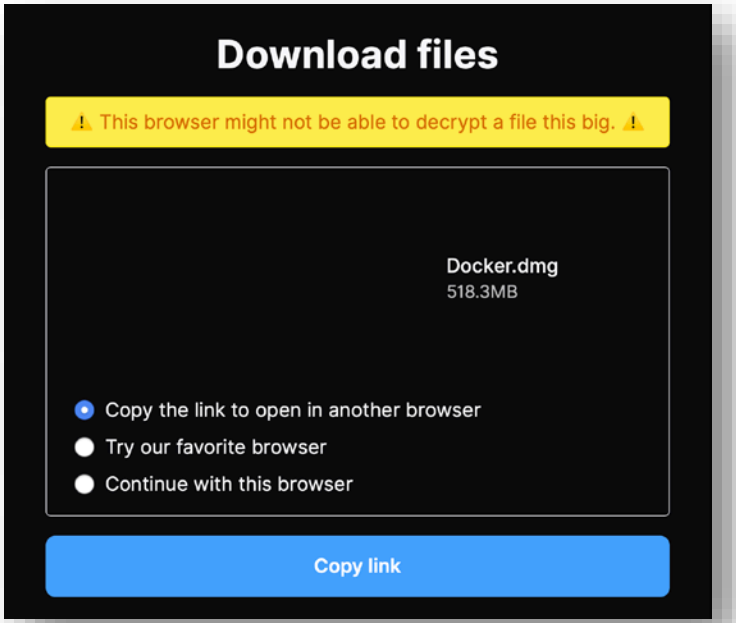
Remember? Creating a ”Save as” Dialog

```
export default async function saveFile(plaintext: ArrayBuffer, fileName: string, fileType:
string) {
  return new Promise((resolve, reject) => {
    const dataView = new DataView(plaintext);
    const blob = new Blob([dataView], { type: fileType });

    const downloadUrl = URL.createObjectURL(blob);
    const a = document.createElement('a');
    a.href = downloadUrl;
    a.download = fileName;
    document.body.appendChild(a);
    a.click();
    URL.revokeObjectURL(downloadUrl);
    setTimeout(resolve, 100);
  });
}
```

GitHub Gist: Downloading an Array Buffer via a "Save as" Dialog in the Browser. https://gist.github.com/thomaskonrad/37772256a86d9f5b0472b3c2440cffee

Supporting Old Browsers

- **Blob download**
  - We can use it for older browsers
  - But we must limit the file size,
  - E.g. max. 256 MB

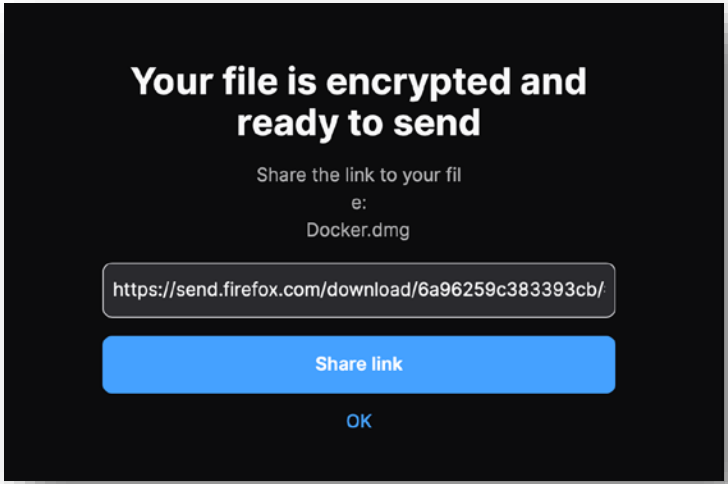


Requirements

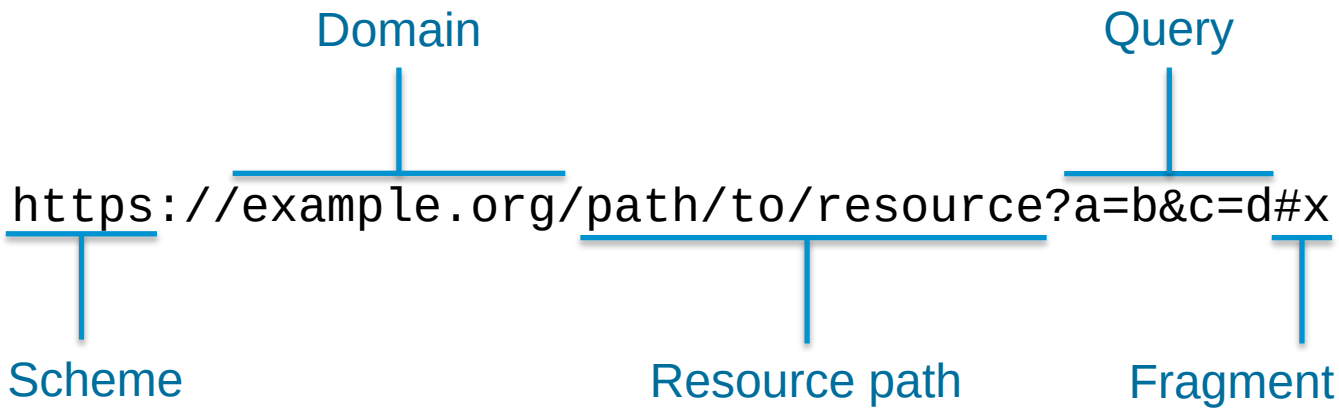
- Ensure confidentiality, integrity, and authenticity ✓
- Encrypt the file in the browser ✓
- Support big files (> 10 GB) ✓
- Support old browsers as well as possible ✓
- Securely distribute keys

Sharing the Master Key

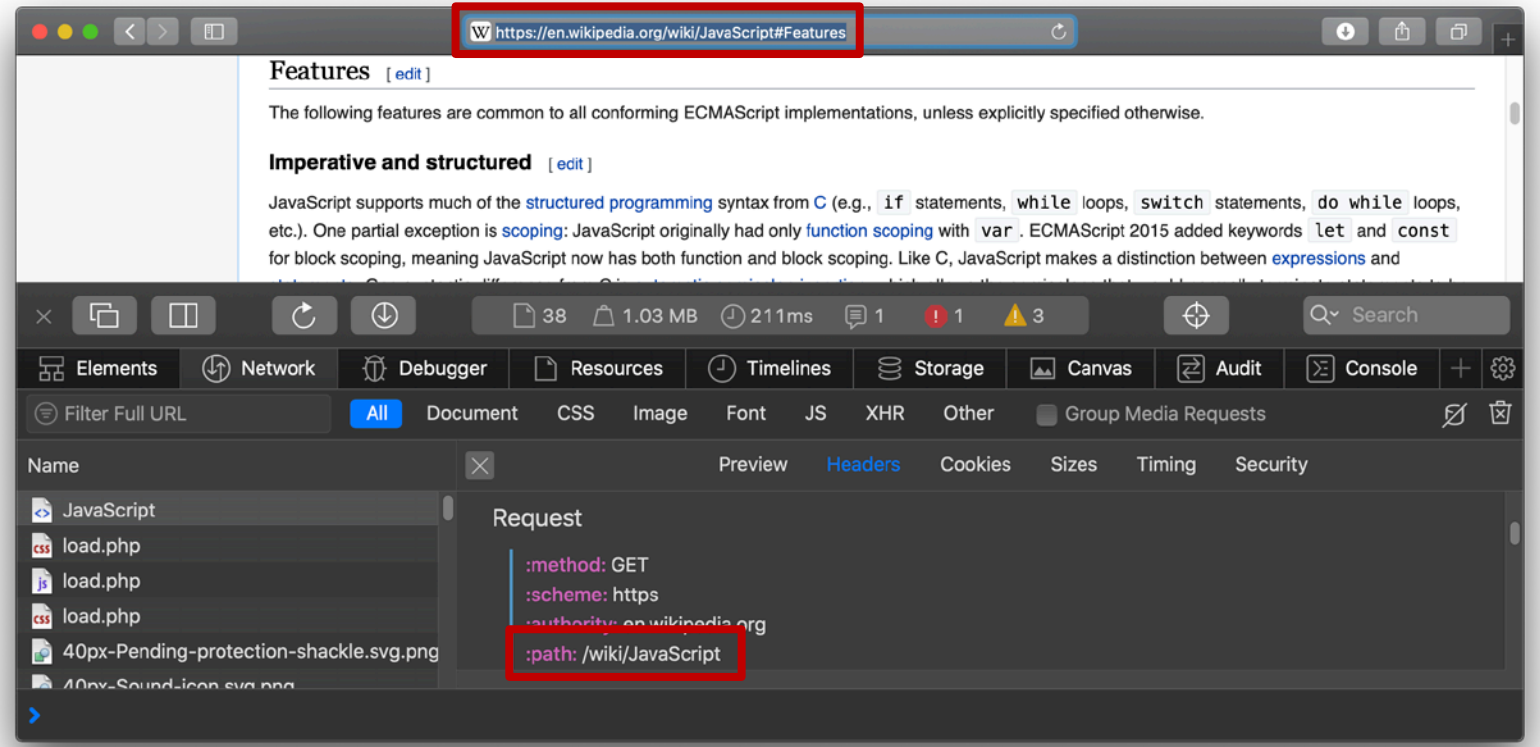
- **Requirements**
  - Share the file via link
  - The key must not be transferred to the server



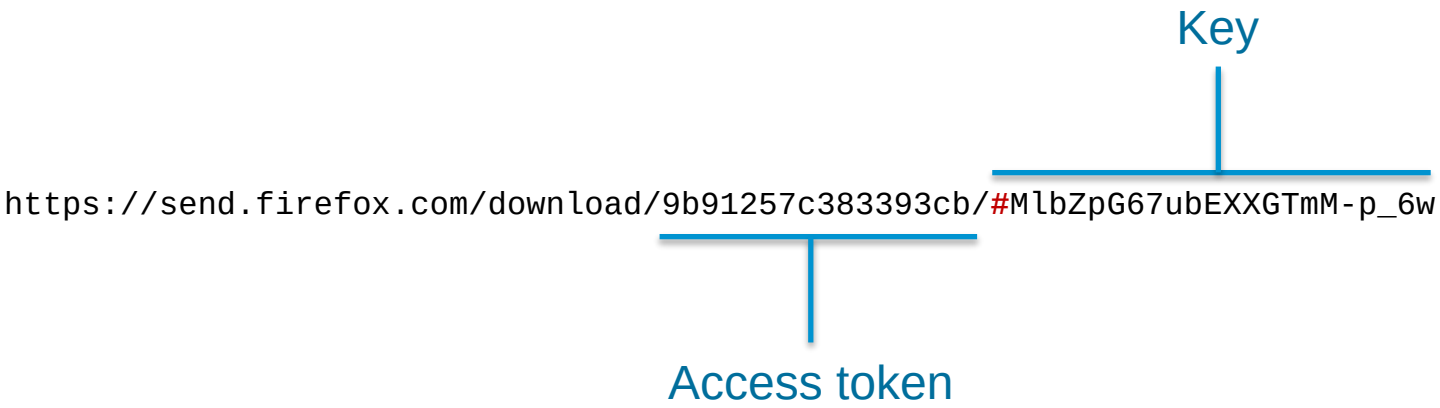
The URL


Domain
Query
https://example.org/path/to/resource?a=b&c=d#x
Scheme
Resource path
Fragment

The URL Fragment Stays in the Browser



A File URL in Firefox Send



Requirements

- Ensure confidentiality, integrity, and authenticity ✓
- Encrypt the file in the browser ✓
- Support big files (> 10 GB) ✓
- Support old browsers as well as possible ✓
- Securely distribute keys ✓

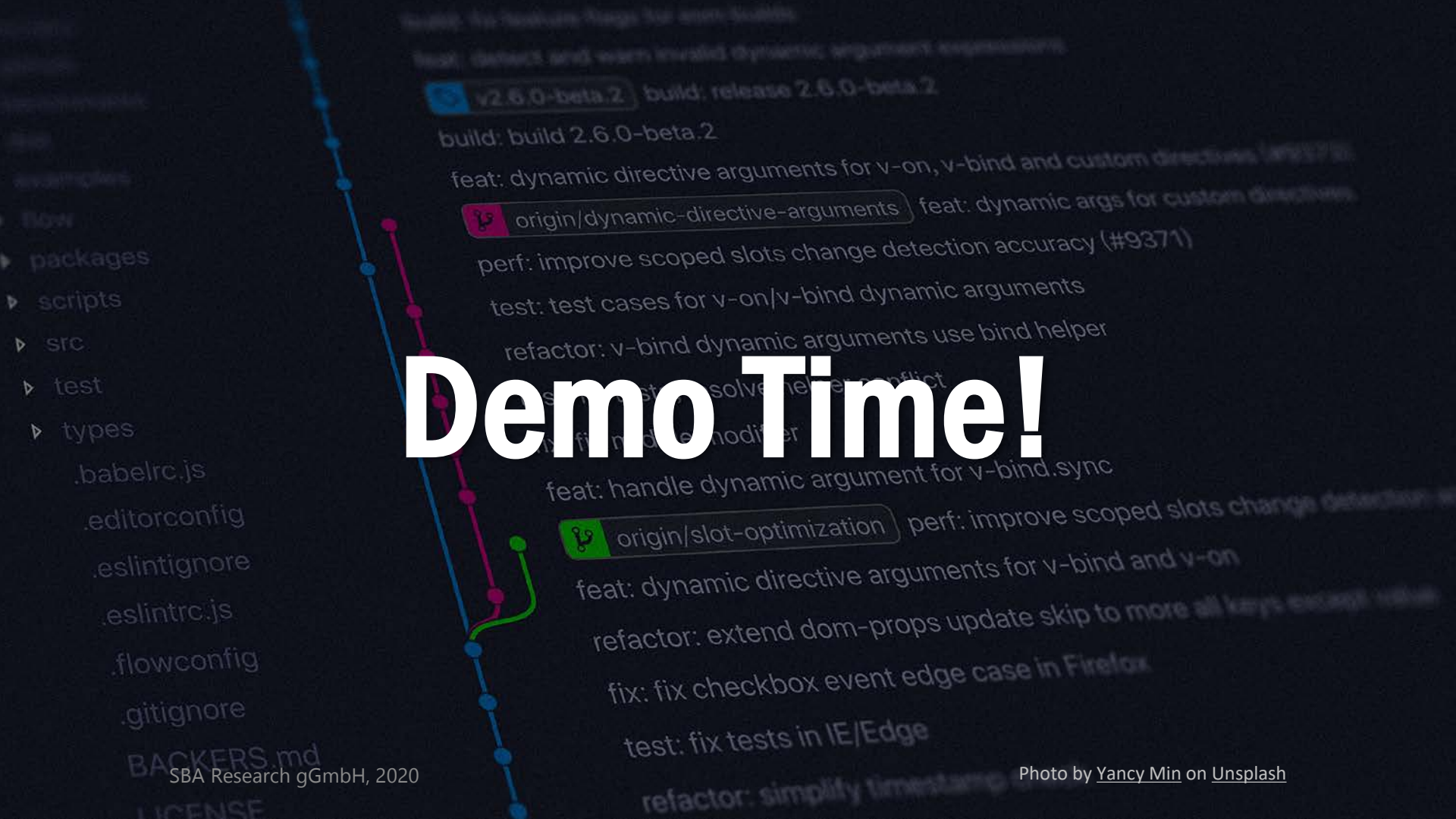


Photo by Yancy Min on Unsplash

Wrapping up

What we have learned today

Wrapping up

- **Ensure confidentiality, integrity, and authenticity**
  - Non-AEAD ciphers don’t guarantee integrity and authenticity
  - Use AEAD ciphers like AES-GCM
  - Use a unique initialization vector (IV) every time you encrypt something

Wrapping up

- **Cryptography in the browser**
  - Modern browsers ship with the Web Crypto API
  - Use `crypto.getRandomValues` to generate random bytes
  - Use `crypto.subtle.importKey`, `crypto.subtle.encrypt` and `crypto.subtle.decrypt` to encrypt data in the browser

Wrapping up

- **Chunked upload**
  - Use the FileReader API to chunk files that you get from an `<input type="file"/>` input
- **Chunked download**
  - Use a Service Worker to intercept requests
  - Use the `ReadableStream` API to chunk and decrypt
  - Support limited size downloads for old browsers via blob download

sec4dev Bootcamp: A Builder's Guide to Single Page Application Security

- **Philippe De Ryck**
- 24 – 26 Feb 2020
- TU Wien, Audimax
- **Learn to build secure SPAs**
  - Cross-site Scripting (XSS)
  - Content Security Policy (CSP)
  - Cross-origin Resource Sharing (CORS)
  - Protecting against malicious third-party content
  - JWT best practices
  - Oauth 2.0 and OpenID Connect
- https://sec4dev.io

Thank you!

Thomas Konrad

SBA Research

tkonrad@sba-research.org

@_thomaskonrad

Slides will be published here!

Photo by Morvanic Lee on Unsplash