

Aplikacja do szyfrowania maili za pomocą kluczy PGP

Autorzy: Tomasz Kusek, Arkadiusz Banaś
Przedmiot: Kryptografia
Uczelnia: Akademia Górnośląska

Cel projektu

Celem projektu jest stworzenie aplikacji, która pozwala na bezpieczne szyfrowanie i deszyfrowanie treści maili przy użyciu kluczy PGP. PGP to sprawdzony standard kryptograficzny, który gwarantuje wysoki poziom bezpieczeństwa komunikacji, dzięki zastosowaniu kluczy asymetrycznych. Aspektem kluczowym projektu jest umożliwienie użytkownikom generowania swoich kluczy PGP na podstawie indywidualnie wybranego hasła, co dodatkowo wzmacnia ochronę ich prywatności.

Projekt koncentruje się na zagadnieniach związanych z kryptografią asymetryczną, w szczególności na algorytmie PGP. Oprogramowanie umożliwia użytkownikom utworzenie bezpiecznego klucza publicznego i prywatnego na podstawie hasła, co stanowi innowacyjne podejście do generowania kluczy PGP.

Wykorzystany algorytm PGP, będący częścią projektu, opiera się na asymetrycznej kryptografii. Po wygenerowaniu zapisuje je w bazie szyfrując dodatkowo algorytmem haszującym md5 na podstawie klucza przypisanego do użytkownika aplikacji, który dodatkowo jest zaszyfrowany szyfrem Bacona.

Projekt zakłada zapewnienie użytkownikom prostego i intuicyjnego narzędzia do bezpiecznej komunikacji e-mailowej. Poprzez wykorzystanie kluczy PGP generowanych na podstawie indywidualnych haseł, projekt stawia na zwiększenie prywatności użytkowników, eliminując jednocześnie konieczność przechowywania kluczy w formie plików.

Bezpieczeństwo: Wykorzystanie kluczy PGP gwarantuje wysoki poziom bezpieczeństwa komunikacji, uniemożliwiając nieautoryzowany dostęp do treści maili. Wszystkie wrażliwe dane w bazie są dodatkowo zaszyfrowane co zapewnia ich bezpieczeństwo w razie wycieku.

Prywatność: Generowanie kluczy na podstawie hasła i przechowywanie je w bazie danych eliminuje potrzebę przechowywania kluczy w formie plików, co zwiększa prywatność użytkowników.

Uniwersalność: Aplikacja umożliwia użytkownikom korzystanie z bezpiecznej komunikacji e-mailowej, niezależnie od używanej platformy czy klienta pocztowego.

Projekt skupia się na praktycznym zastosowaniu teorii kryptografii w celu stworzenia efektywnego narzędzia, które może być używane przez szerokie grono użytkowników do zabezpieczenia swojej prywatnej korespondencji elektronicznej.

Zastosowane algorytmy szyfrujące

Pretty Good Privacy (PGP) to kompleksowy system szyfrowania stosowany do ochrony poufności danych, zwłaszcza w kontekście komunikacji elektronicznej. Opracowany przez Phila Zimmermanna w 1991 roku, PGP wykorzystuje zaawansowane techniki kryptograficzne, w tym asymetryczne klucze publiczny i prywatny, funkcje skrótu, oraz podpisy cyfrowe.

Asymetryczne Klucze:

Klucz Publiczny: Służy do szyfrowania danych i jest dostępny publicznie.

Klucz Prywatny: Wykorzystywany jest do odszyfrowywania danych i musi być trzymany w tajemnicy.

Funkcje Skrótu:

PGP używa funkcji skrótu (np. SHA-256) do generowania podpisów cyfrowych, co umożliwia weryfikację integralności przesyłanych danych.

Podpisy Cyfrowe:

PGP pozwala na podpisywanie cyfrowe, co potwierdza autentyczność nadawcy i niemożność zaprzeczenia wysłania danej wiadomości.

Historia PGP zaczęła się jako reakcja na rosnące obawy o prywatność w erze cyfrowej. Phil Zimmermann stworzył PGP jako narzędzie, które pozwoliłoby jednostkom na ochronę prywatności w komunikacji elektronicznej. Opublikowanie PGP jako darmowego oprogramowania, dostępnego publicznie, przyczyniło się do rozwoju standardów otwartego źródła w dziedzinie kryptografii.

Założenia PGP obejmują:

Prywatność i Bezpieczeństwo: Głównym celem PGP jest zapewnienie prywatności i bezpieczeństwa komunikacji elektronicznej, uniemożliwiając dostęp nieupoważnionym osobom do treści przesyłanych danych.

Otwarty Model: PGP przyjmuje otwarty model, co oznacza, że kod jest dostępny publicznie, co pozwala na niezależne weryfikowanie bezpieczeństwa algorytmu.

Szyfr Bacona, znany również jako szyfr Franciszka Bacona, to technika szyfrowania, w której litery są zamieniane na ciągi liter A i B. Każda litera alfabetu jest reprezentowana przez unikalny ciąg pięciu znaków A i B (np. AAAAA, AAAAB, AABBB, itd.).

Działanie:

Każdej literze alfabetu przypisany jest unikalny ciąg pięciu znaków A i B.

Tekst do zaszyfrowania jest zamieniany na ciągi A i B zgodnie z przypisanymi regułami.

Każdy ciąg reprezentuje jedną literę.

Odszyfrowanie polega na odnalezieniu odpowiadającej każdemu ciągowi litery alfabetu.

Funkcja szyfrująca klucz szyfrowania szyfrem Bacona:

```
const baconEncrypt = (key: string): string => {
  const baconCipher: { [key: string]: string } = {
    'a': 'AAAAA', 'b': 'AAAAB', 'c': 'AAABA', 'd': 'AAABB', 'e': 'AABAA',
    'f': 'AABAB', 'g': 'AABBA', 'h': 'AABBB', 'i': 'ABAAA', 'j': 'ABAAB',
    'k': 'ABABA', 'l': 'ABABB', 'm': 'ABBAA', 'n': 'ABBAB', 'o': 'ABBBA',
    'p': 'ABBBB', 'q': 'BAAAA', 'r': 'BAAAB', 's': 'BAABA', 't': 'BAABB',
    'u': 'BABAA', 'v': 'BABAB', 'w': 'BABBA', 'x': 'BABBB', 'y': 'BBAAA',
    'z': 'BBAAB', ' ': 'BBBBB'
  };

  return key.split('').map((char: string) => baconCipher[char.toLowerCase()]).join('');
};
```

Algorytm MD5 to algorytm funkcji skrótu kryptograficznego, który generuje 128-bitową wartość skrótu, zwaną "hashem". Został stworzony przez Rona Rivesta. Jest powszechnie używany do sprawdzania integralności danych oraz w systemach uwierzytelniania.

Działanie:

Inicjalizacja Stanu: Inicjalizacja 128-bitowego stanu początkowego.

Przetwarzanie Bloków Danych: Wiadomość jest podzielona na bloki 512 bitów, które są przetwarzane sekwencyjnie.

Dodawanie Paddingu: Jeśli wiadomość nie jest wielokrotnością 512 bitów, zostaje dodany padding (doprowadzający do wielokrotności 512).

Dodawanie Długości: Do końca wiadomości dodawany jest blok 64-bitowy zawierający długość oryginalnej wiadomości.

Operacje Logiczne: Kolejne operacje logiczne (rotacje, dodawania modulo) wykonywane są na blokach danych.

Aktualizacja Stanu: Stan jest aktualizowany po przetworzeniu każdego bloku.

Generowanie Skrótu: Ostateczny 128-bitowy hash jest generowany.

Projekt aplikacji

Do korzystania z aplikacji wymagane jest aby użytkownik posiadał konto. Aplikacja umożliwia generowanie kluczy PGP na podstawie indywidualnie wprowadzonego hasła, co pozwala użytkownikowi skoncentrować się na bezpieczeństwie swoich kluczy. Dodatkowo, aplikacja zapisuje klucze PGP w bazie, dzięki czemu użytkownik nie musi ich przechowywać. Głównym celem aplikacji jest umożliwienie użytkownikowi wysyłki zaszyfrowanych maili, gdzie treść maila jest dostępna jedynie po wprowadzeniu hasła, które było używane do generowania kluczy.

W naszej aplikacji podeszliśmy do kluczy PGP nieco inaczej. Aby zapewnić bezpieczeństwo oraz wygodę użytkownika zdecydowaliśmy, że klucze będą przechowywane w bazie a odbiorca takiej wiadomości będzie otrzymywał jedynie maila z linkiem. Po kliknięciu w link zostaje przekierowany na stronę, na której musi wprowadzić hasło podane podczas wysyłki maila. Jeżeli hasło jest prawidłowe, następuje proces deszyfrowania wiadomości za pomocą klucza prywatnego PGP. Jest to trochę odmienne zastosowanie kluczy PGP ale dzięki temu osoby mniej obeznane w IT będą mogły swobodnie zabezpieczyć swoją elektroniczną korespondencję.

Opisując naszą aplikację, można wskazać kilka kluczowych cech i zalet, które wyróżniają ją na tle innych rozwiązań z zakresu bezpiecznej komunikacji elektronicznej:

Rejestracja konta i generowanie kluczy PGP na podstawie hasła:

Aby korzystać z aplikacji, użytkownik musi założyć konto, co pozwala na personalizację i zabezpieczenie dostępu do funkcji.

Generowanie kluczy PGP na podstawie indywidualnie wprowadzonego hasła dodaje warstwę dodatkowego bezpieczeństwa, umożliwiając użytkownikowi skupienie się na bezpieczeństwie swoich kluczy.

Przechowywanie kluczy PGP w bazie danych:

Zapisywanie kluczy PGP w bazie danych eliminuje konieczność przechowywania kluczy przez użytkownika, co może być trudne i niewygodne. Wprowadza to element wygody dla użytkownika.

Wysyłka zaszyfrowanych maili:

Głównym celem aplikacji jest umożliwienie użytkownikowi wysyłki zaszyfrowanych maili, co przyczynia się do zwiększenia prywatności i bezpieczeństwa komunikacji elektronicznej.

Bezpieczny sposób udostępniania wiadomości:

Odbiorca otrzymuje jedynie maila z linkiem, co dodatkowo zabezpiecza zawartość wiadomości przed przypadkowym dostępem lub przeglądaniem przez niepowołane osoby.

Weryfikacja hasła przy odbiorze:

Proces deszyfrowania wiadomości wymaga wprowadzenia hasła podanego podczas wysyłki maila. To dodatkowe zabezpieczenie, które zapewnia, że tylko osoba mająca odpowiednie hasło może odczytać zawartość wiadomości.

Ułatwienia dla osób mniej obeznanych z IT:

Inne podejście do wykorzystania kluczy PGP, poprzez zapisanie ich w bazie i umożliwienie deszyfrowania za pomocą hasła, może przyciągnąć osoby mniej zaznajomione z zagadnieniami technicznymi, zachęcając je do korzystania z bezpiecznych metod komunikacji elektronicznej.

Ograniczenie dostępu do treści wiadomości:

Treść maila jest dostępna jedynie po wprowadzeniu prawidłowego hasła, co dodatkowo zabezpiecza informacje przed nieautoryzowanym dostępem.

Podsumowując, aplikacja ta łączy w sobie bezpieczeństwo kluczy PGP z prostotą obsługi, kierując się potrzebami użytkowników, którzy poszukują skutecznych, a jednocześnie łatwych do zrozumienia narzędzi do ochrony swojej elektronicznej korespondencji.

Zastosowane biblioteki:

- Crypto - funkcje Szyfrujące i Deszyfrujące: Biblioteka crypto dostarcza zestaw funkcji kryptograficznych, które są używane do szyfrowania i deszyfrowania danych. W kontekście aplikacji, może być wykorzystywana do obsługi różnych algorytmów szyfrowania. Służy do generowania skrótu (hashu) zaszyfrowanego klucza przed zapisem w bazie danych. Zastosowanie w aplikacji: szyfrowanie i deszyfrowanie wrażliwych danych.

```
// Algorytm szyfrowania AES-256-CBC.
const algorithm: string = 'aes-256-cbc';
// Funkcja szyfrująca dane z użyciem szyfru Bacona
export const decrypt = (text: string, key?: string, iv_length?: number) => {
  key = key || ''
  if (text == null || text == '' || typeof text == 'undefined')
    return text

  key = crypto.createHash('md5').update(baconEncrypt(key), 'utf-8').digest('hex').toUpperCase();
  let length = iv_length || 16
  let iv: Buffer = Buffer.alloc(length)
  let decipher = crypto.createDecipheriv(algorithm, key, iv)

  let decrypted = decipher.update(text, 'hex', 'utf8') + decipher.final('utf8')
  return decrypted
};

export const encrypt = (text: string, key?: string, iv_length?: number) => {
  key = key || ''
  if (text == null || text == '' || typeof text == 'undefined')
    return text
  if (typeof text == 'number')
    text = text + ''
  key = crypto.createHash('md5').update(baconEncrypt(key), 'utf-8').digest('hex').toUpperCase();
```

```

let length = iv_length || 16
let iv = Buffer.alloc(length)
let cipher = crypto.createCipheriv(algorithm, key, iv);
let encrypted = cipher.update(text, 'utf8', 'hex') + cipher.final('hex');

return encrypted
};

```

- Openpgp - protokół PGP: Biblioteka openpgp jest kluczowa dla realizacji funkcji związanych z protokołem PGP. Pozwala na generowanie kluczy publicznych i prywatnych, a także na szyfrowanie i deszyfrowanie treści maili w standardzie PGP. Zastosowanie w aplikacji: funkcje generujące klucze.

```

import * as openpgp from 'openpgp';

export const decrypt = async (message: string, password: string, privateKeyArmored: string,
publicKeyArmored: string) => {
  const publicKey = await openpgp.readKey({ armoredKey: publicKeyArmored });

  const privateKey = await openpgp.decryptKey({
    privateKey: await openpgp.readPrivateKey({ armoredKey: privateKeyArmored }),
    passphrase: password
  });

  const encryptedMessage = await openpgp.readMessage({
    armoredMessage: message // parse armored message
  });
  const { data: decrypted } = await openpgp.decrypt({
    message: encryptedMessage,
    verificationKeys: publicKey as any, // optional
    decryptionKeys: privateKey as any
  });
  return decrypted;
}

export const encrypt = async (message: string, password: string, privateKeyArmored: string,
publicKeyArmored: string,) => {
  try {
    const publicKey = await openpgp.readKey({ armoredKey: publicKeyArmored });

    const privateKey = await openpgp.decryptKey({
      privateKey: await openpgp.readPrivateKey({ armoredKey: privateKeyArmored }),
      passphrase: password
    });

    const encrypted = await openpgp.encrypt({
      message: await openpgp.createMessage({ text: message }), // input as Message
      encryptionKeys: publicKey,
      signingKeys: privateKey // optional
    });
    return encrypted;
  } catch (err) {

```

```

    return { err }
  }
}

```

- Bcrypt - Funkcje Hashujące z Dodatkowym Sparаметryzowaniem: bcrypt jest używane do bezpiecznego haszowania haseł użytkowników podczas rejestracji. Zapewnia dodatkowe zabezpieczenia poprzez parametryzację funkcji haszującej, co utrudnia ataki typu brute-force. Zastosowanie w aplikacji: funkcje haszujące i sprawdzające poprawność haseł.

```

const compareRes = await bcrypt.compare(opts.password, dbUser.password || '')
const passwordHash = await bcrypt.hash(opts.password, 12)

```

- Nodemailer - Obsługa Poczty Elektronicznej: Biblioteka nodemailer umożliwia aplikacji wysyłanie e-maili. Jest stosowana do wysyłania maili z linkami do zaszyfrowanych treści, zgodnie z wymaganiami projektu. Zastosowanie w aplikacji: narzędzie do wysyłania maili.

```

const transporter = nodemailer.createTransport({
  host: mailConfig.host || mailConfig.user || process.env.MAILER_HOST,
  port: mailConfig.port || process.env.MAILER_SMTP_PORT,
  secure: typeof (mailOptions.ssl) !== undefined ? mailOptions.ssl :
(process.env.MAIL_SSL == '1'),
  requireTLS: false,
  auth: {
    user: mailConfig.user || process.env.MAILER_USER,
    pass: mailConfig.password || process.env.MAILER_PASS
  },
  logger: false
});
transporter.use('compile', inlineBase64({ cidPrefix: 'img_' }));

transporter.sendMail(options, function (err, info) {
  transporter.close()
  if (err) {
    Logger.error(err)
    return resolve({ err })
  }
  return resolve({ ...info, from: options.from.address })
})

```

- Mysql - Komunikacja z Bazą Danych: Biblioteka ta umożliwia skuteczną obsługę bazy danych MySQL. Działa jako interfejs pomiędzy aplikacją a bazą danych, umożliwiając operacje takie jak wstawianie, pobieranie, aktualizowanie i usuwanie danych. Zastosowanie w aplikacji: inicjalizacja puli połączenia z bazą danych, wykonywanie zapytań, escapowanie danych (chroni przed sql injection).

```

import { createPool, Pool, escape as dbEscape } from 'mysql';

let pool: Pool;

/**
 * generates pool connection to be used throughout the app

```



```

    */
export const init = () => {
    try {
        pool = createPool({
            connectionLimit: 5,
            host: process.env.DB_HOST,
            user: process.env.DB_USER,
            password: process.env.DB_PASSWORD,
            database: process.env.DB_DATABASE,
        });

        console.debug('MySQL Adapter Pool generated successfully for database: ',
process.env.DB_DATABASE);
    } catch (error) {
        console.error('[mysql.connector][init][Error]: ', error);
        throw new Error('failed to initialized pool');
    }
};

/**
 * executes SQL queries in MySQL db
 *
 * @param {string} query - provide a valid SQL query
 * @param {string[] | Object} params - provide the parameterized values used
 * in the query
 */
export const execute = <T>(query: string, params: string[] | Object, findOne: boolean | any):
Promise<T> => {
    try {
        if (!pool) throw new Error('Pool was not created. Ensure pool is created when running
the app.');
```

```

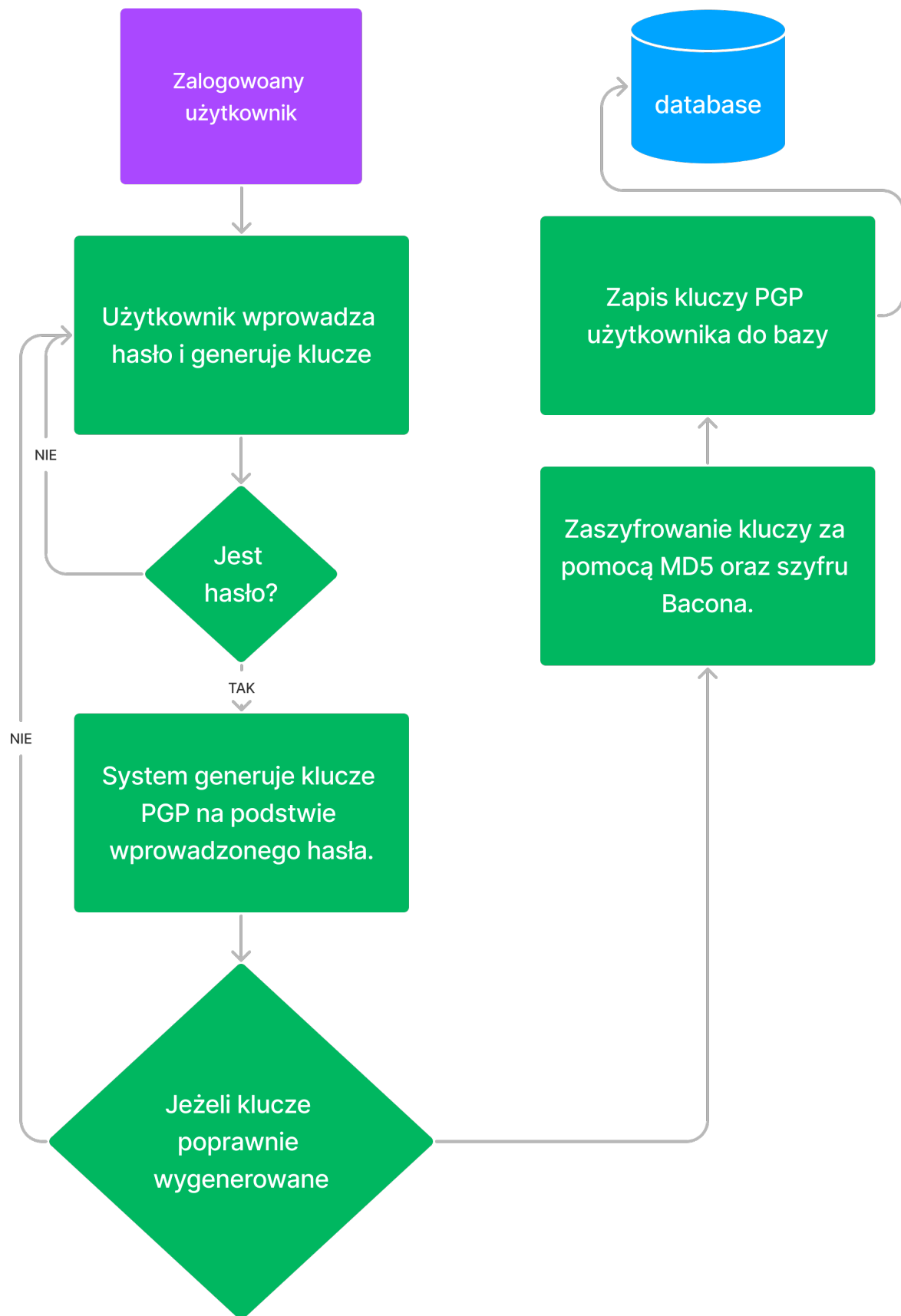
        return new Promise<T>((resolve, reject) => {
            pool.query(query, params, (error, results) => {
                if (error) return reject(error);
                if (findOne)
                    if (results?.length > 0) results = results[0]
                    else results = {}
                return resolve(results);
            });
        });

    } catch (error) {
        console.error('[mysql.connector][execute][Error]: ', error);
        throw new Error('failed to execute MySQL query');
    }
}

export const escape = dbEscape;

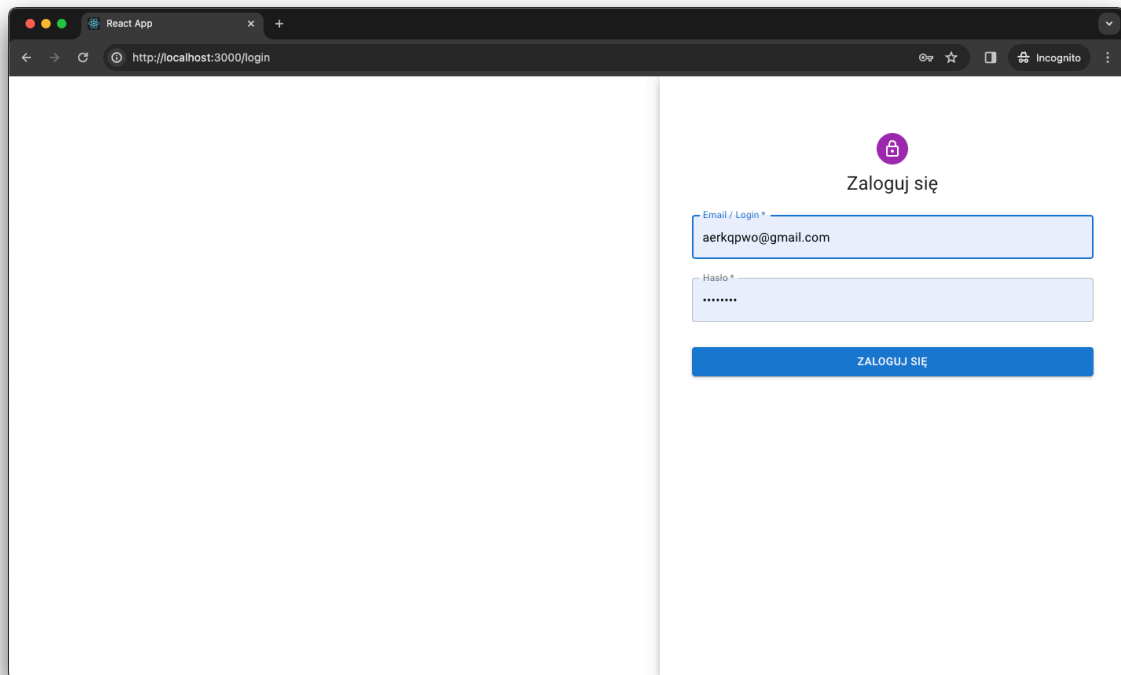
```

Proces generowania kluczy



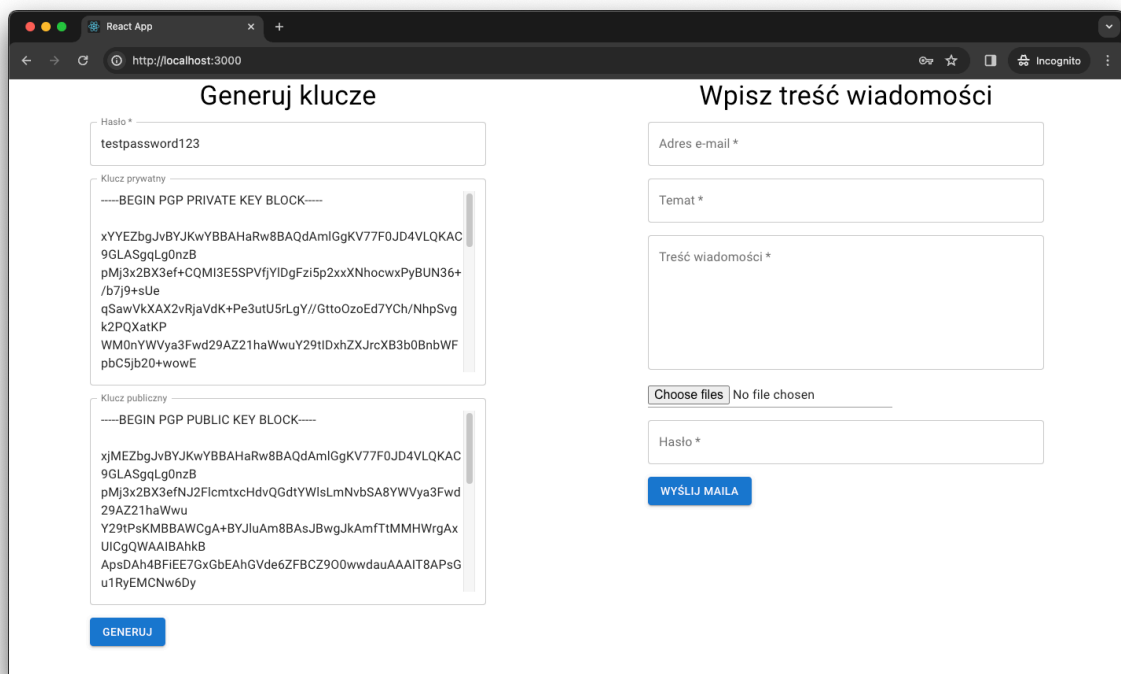
Działanie aplikacji

1. Logowanie



The screenshot shows a web browser window with the address bar displaying "http://localhost:3000/login". The page has a white background and a purple lock icon at the top center. Below the icon, the text "Zaloguj się" is displayed. There are two input fields: "Email / Login *" with the value "aerkqpwo@gmail.com" and "Hasło *" with masked characters "*****". A blue button labeled "ZALOGUJ SIĘ" is positioned below the input fields.

2. Generowanie kluczy



The screenshot shows a web browser window with the address bar displaying "http://localhost:3000". The page is divided into two main sections. The left section, titled "Generuj klucze", contains a "Hasło *" field with the value "testpassword123". Below it, there are two text areas for generated keys. The first is labeled "Klucz prywatny" and contains a long string of characters starting with "-----BEGIN PGP PRIVATE KEY BLOCK-----". The second is labeled "Klucz publiczny" and contains a long string of characters starting with "-----BEGIN PGP PUBLIC KEY BLOCK-----". A blue button labeled "GENERUJ" is at the bottom of this section. The right section, titled "Wpisz treść wiadomości", contains three input fields: "Adres e-mail *", "Temat *", and "Treść wiadomości *". Below these fields is a file selection button labeled "Choose files" with the text "No file chosen". A "Hasło *" field is also present. A blue button labeled "WYŚLIJ MAILA" is at the bottom of this section.

3. Wysyłka wiadomości

React App

http://localhost:3000

Generuj klucze

Hasło *

Klucz prywatny

-----BEGIN PGP PRIVATE KEY BLOCK-----

```
xYyEZbgJvBYJKwYBBAHaRw8BAQdAmlGgKV77F0JD4VLQKAC
9GLASgqLg0nzB
pMj3x2BX3ef+CQMI3E5SPVfYlDgFzi5p2xxXNhocwxPyBUN36+
/b7j9+sUe
qSawVkXAX2vRjaVdK+Pe3utU5rLgY//GttoOzoEd7YCh/NhpSvg
k2PQXatKP
WM0nyWVya3Fwd29AZ21haWwuY29tIDxhZXJrcXB3b0BnbWF
pbC5jb20+wowE
```

Klucz publiczny

-----BEGIN PGP PUBLIC KEY BLOCK-----

```
xjMEZbgJvBYJKwYBBAHaRw8BAQdAmlGgKV77F0JD4VLQKAC
9GLASgqLg0nzB
pMj3x2BX3efNJ2FcmxcHdvQGdtYwIsLmNvbSA8YVWya3Fwd
29AZ21haWwu
Y29tPsKMBBAWCgA+BYJluAm8BASjBwgJkAmFTtMMHWRgAx
UICgQWAAIBAhkB
ApsDAh4BFIEE7GxGbEAhGVde6ZFBZCZ900wwdauAAAIT8APsG
u1RyEMCNw6Dy
```

GENERUJ

Wpisz treść wiadomości

Adres e-mail *

aerkqpw@gmail.com

Temat *

Zaszyfrowana wiadomość

Treść wiadomości *

Zaszyfrowana treść wiadomości ;)

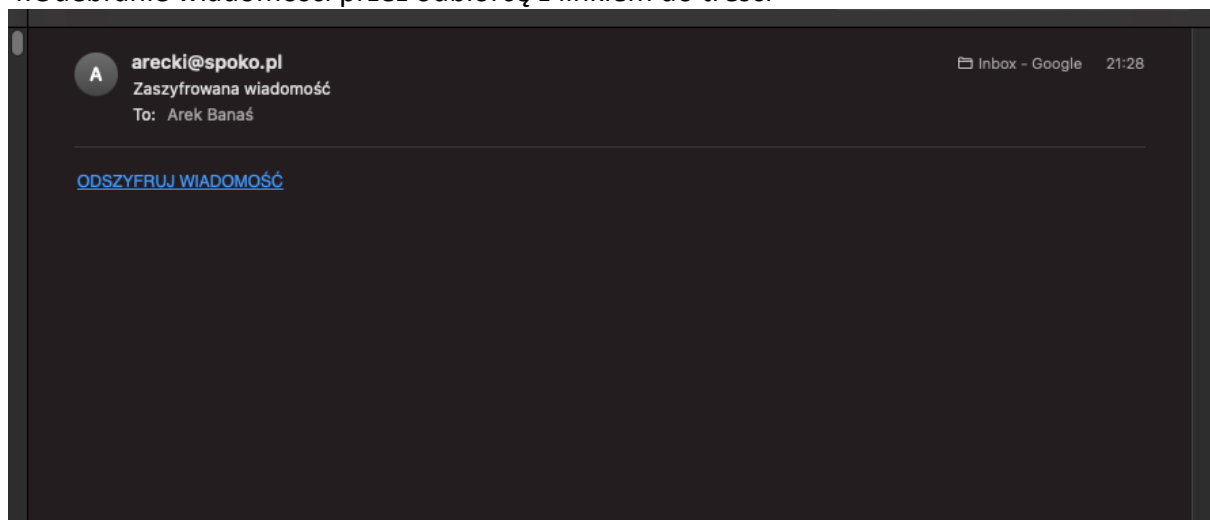
Choose files No file chosen

Hasło *

testpassword123

WYŚLIJ MAILA

4. Odebranie wiadomości przez odbiorcę z linkiem do treści



5. Wprowadzenie hasła

React App

React App

http://localhost:3000/decrypt/142

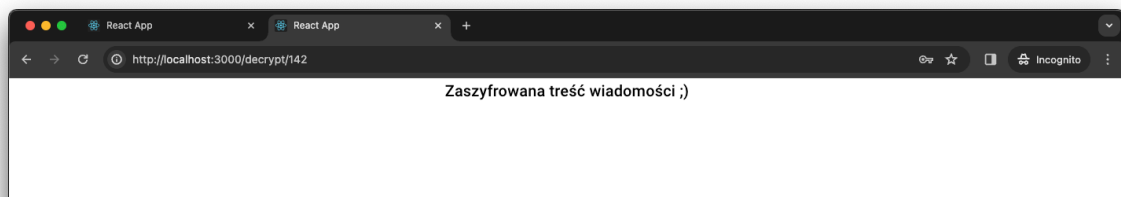
Rozszyfruj wiadomości

Hasło *

testpassword123

ROZSZYFRUJ

6.Odszyfrowanie wiadomości



Podsumowanie

Problemy:

Naszym jedynym problemem, który nas spotkał w procesie tworzenia aplikacji to błąd użycia kluczy PGP z innych generatorów dostępnych w internecie.

Co dalej?

Projekt ma wiele możliwości rozwoju. Naszym pomysłem na dalszy rozwój jest dodanie książki kontaktowej, która umożliwi zapisywanie kluczy publicznych odbiorców, do których chcemy wysyłać wiadomości. Aplikacji automatycznie zapisze odbiorcę razem z kluczem w książce kontaktowej podczas wysyłki.

Co się udało a co nie?

Udało nam się:

- szyfrować i deszyfrować maile za pomocą kluczy PGP
- szyfrować i deszyfrować wrażliwe dane
- wysyłać maile z serwera pocztowego użytkownika

Nie udało nam się:

- uzyskać kompatybilności z innymi generatorami kluczy PGP