

JavaScriptによるEnd-to-Endセキュリティ

第4回 データの真正性・本人確認のためのテクニック 編

栗原 淳

2019年10月31日

はじめに

はじめに

第 1,2,3 回では

- End-to-End (E2E) セキュリティの原則と必要性
- JavaScript で AES を使った暗号化のお作法
- JavaScript で公開鍵暗号 (RSA/楕円曲線) を使った暗号化のお作法

を勉強した。

今回は、第 3 回の最後に懸案事項だった

「データのやり取りしてる相手って本当に正しい相手？」
を保証する方法を学んでいく。

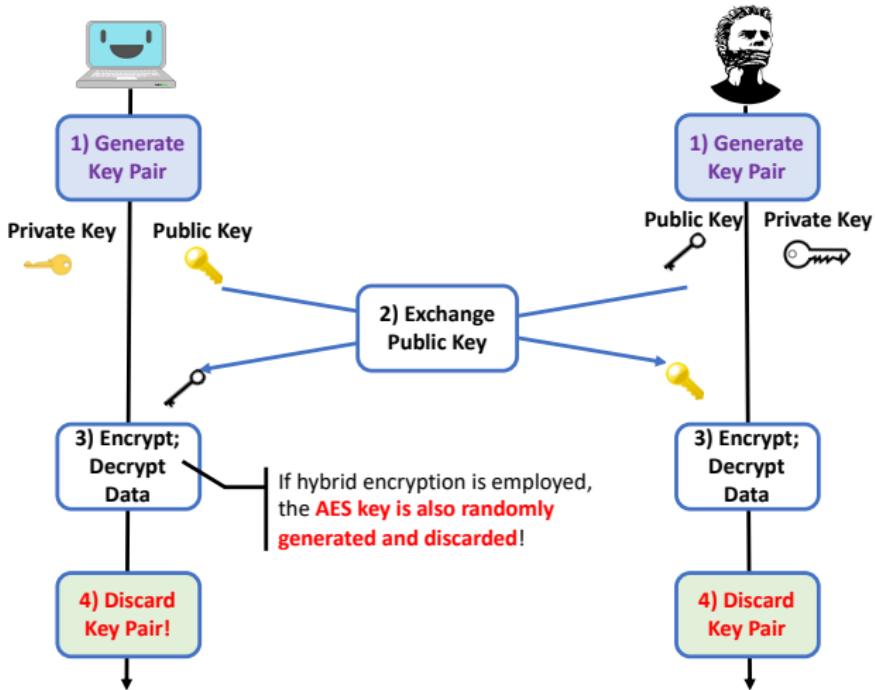
第3回のおさらい: Ephemeral Scheme

公開鍵暗号化の Ephemeral Scheme での運用

公開鍵・秘密鍵ペアを都度生成、1回限りで使い捨てることで、
Perfect Forward Secrecy¹ を担保する運用方法。

Perfect Forward Secrecy を守り、End-to-End 暗号化の強固な運用を。

¹長期的に保存されているマスター秘密鍵の漏洩や、一部の暗号化データがクラックされたとしても、それ以外の過去に暗号化されたデータは復号されてしまうことはないという概念。



Ephemeral Scheme のイメージ

まずははじめに、「送ってきた Ephemeral な公開鍵は、本当に自分がやりとりしたい相手の公開鍵か？」の確認が必須。²

²意図しない相手の公開鍵で暗号化して機密データを漏らさぬように、ということ。

というわけで、「E2E で安全にデータをやり取りする」ための基礎部分の最後のピースを今日は学ぶ。

この講義で最終的に学びたいこと

- 本人確認やデータの改ざん防止を担保する方法
 - データ毎に固有の指紋を生成する「**ハッシュ**」
 - 「共通鍵」を使った改ざん防止方法「**MAC**」³
 - 「公開鍵」を使った本人確認・改ざん防止方法「**電子署名**」⁴
- そしてその具体的な JavaScript での実装方法・お作法

細かい話もするが、数式は使わない。

「イメージ」と「コードの流れ＆その流れの必要性」をつかめるようにする。

³HMAC (RFC2104 <https://tools.ietf.org/html/rfc2104>)

⁴RSASSA PKCS#1-v1.5/PSS (PKCS#1 RFC8017 <https://tools.ietf.org/html/rfc8017>), ECDSA (FIPS PUB186-4 <https://csrc.nist.gov/publications/detail/fips/186/4/final>)

栗原 淳 (Jun Kurihara)

- (株) ゼタント 主任研究員
(株) 国際電気通信基礎技術研究所 (ATR) 連携研究員
- 博士 (工学),
専門: セキュリティ、応用数学、システムアーキテクチャとか
- Web システム (フロントエンド・バックエンド) を作ったり、
論文他のアルゴリズムを実装したり、研究して論文書いたり、
セキュリティ技術中心に手広くやってます。
- GitHub: <https://github.com/junkurihara>
LinkedIn: <https://www.linkedin.com/in/junkurihara>

この講義の対象と事前準備

対象:

- 暗号・セキュリティ技術に興味がある初学者
- Web に暗号技術を導入したい Web 系のエンジニア

必須ではないが触って楽しむのには必要な事前準備:

- Bash, Git が使えるようになっていること
- Node.js, npm, yarn が使えるようになっていること
- Google Chrome 系ブラウザ and/or Firefox が利用可能のこと

今後の予定 (暫定)

- 1 導入&JS の暗号化コードを触ってみる
- 2 AES を正しく・安全に暗号化するには？
- 3 公開鍵暗号はどうやって使う？その使い方のコツは？
- 4 ハッシュ・MAC・署名、それぞれの使い所と使い方は？ ← 今日はココ
- 5 RFC にまつわるあれこれ（証明書・鍵フォーマット・etc...）

「こういうのを知りたい」というリクエストがあれば是非。
マニアックすぎて最後の RFC の話題はやるかどうか未定。

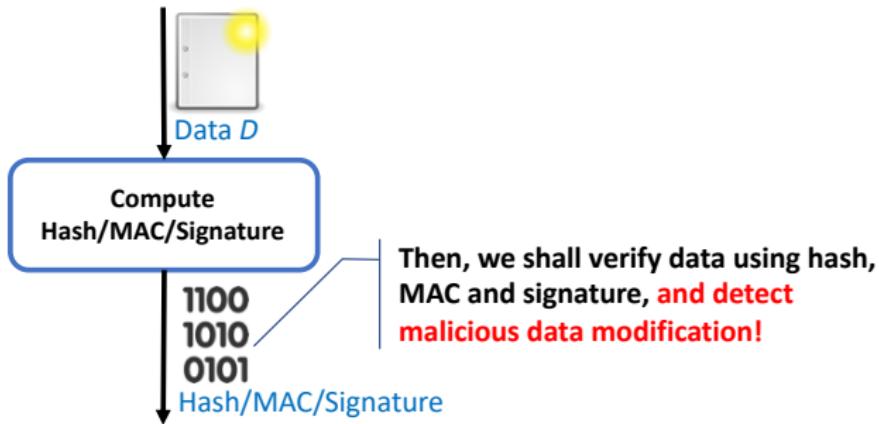
セカンドシーズンも検討中。⁵

⁵場所等変えてもっと来やすい場所へ。。。

サンプルコードの準備

準備

説明を聞きつつ手を動かすため、まず環境準備。今回は、JavaScript (Node.js) を使って手元でデータの Hash/MAC/署名をいじってみる。そしてその効果を実感する。



※サンプルコードはブラウザでも動く。

src/commands-browser.html を開くとこれから Node.JS で試すデモが開発者コンソールで実行される。適宜試したり比較すると良い。

※前回のコードの公開鍵に署名をつけたりして Ephemeral Scheme を作ってみると良い。

環境

以下の環境が前提:

- Node.js (> v10) がインストール済。yarn が使えること。⁶
- ブラウザとして、Google Chrome (系ブラウザ)、もしくは Firefox がインストール済み
- Visual Studio Code や WebStorm などの統合開発環境がセットアップ済みだとなお良い。

⁶ インストールコマンド: `npm i -g yarn`

JavaScript プロジェクトの準備

1 プロジェクトの GitHub リポジトリ⁷ を Clone

```
$ git clone https://github.com/zettant/e2e-security-04  
$ cd e2e-security-04/sample
```

2 依存パッケージのインストール

```
$ yarn install
```

3 ライブラリのビルド

```
$ yarn build
```

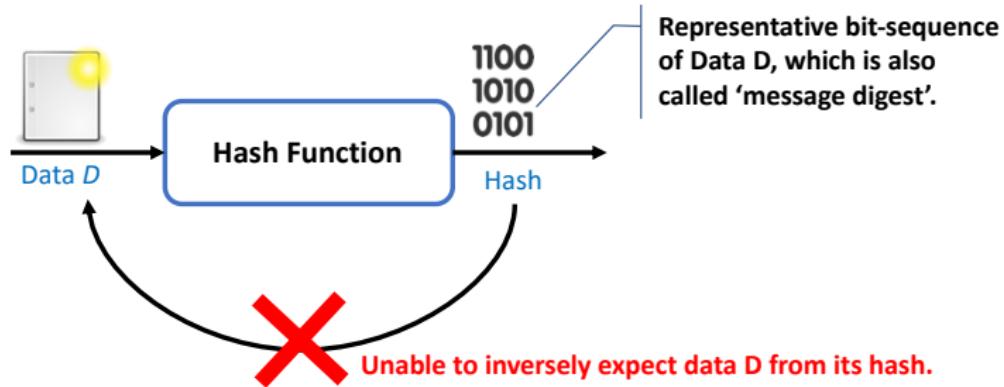
⁷<https://github.com/zettant/e2e-security-04>

データの指紋: Hash

Hash および Hash 関数とは

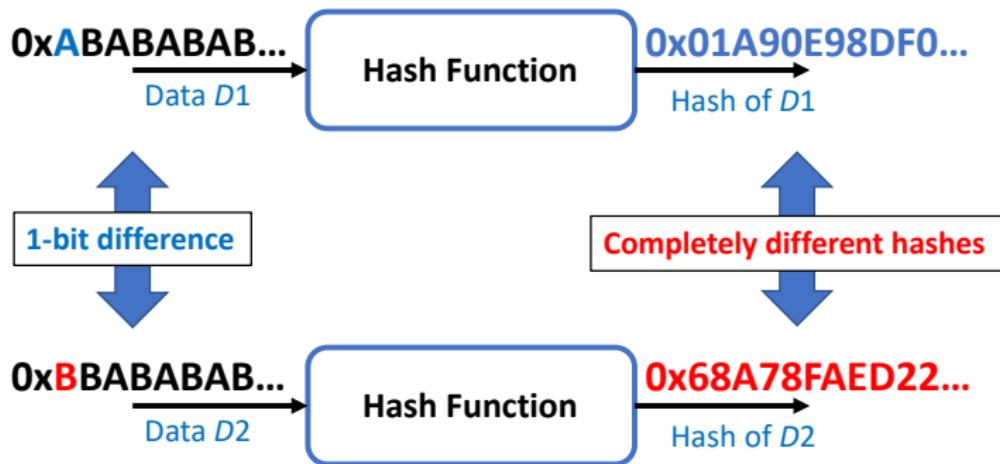
Hash および Hash 関数

あるデータに対し、そのデータを「代表するビット列」を計算する不可逆の関数を「Hash 関数」。導出したビット列を「Hash」⁸と呼ぶ。



⁸あるいは Hash 値、Message Digest

1ビットでもデータが異なれば、全く違うHashが導出される。



Hash および Hash 関数の役割

同じくデータ固有のビット列を導出する Checksum と似ているが、その用途はより強力で多岐にわたる。

■ Checksum

- 通信路上などでのデータの(偶発的な)エラー検知

⇒ データから一意に導ける値・高速な処理が可能なことが必須

■ Hash

- データのエラー・改ざん検知
- Hash をデータ実態の代替として署名を生成
- 多数のデータの索引作成⁹
- データの重複検出

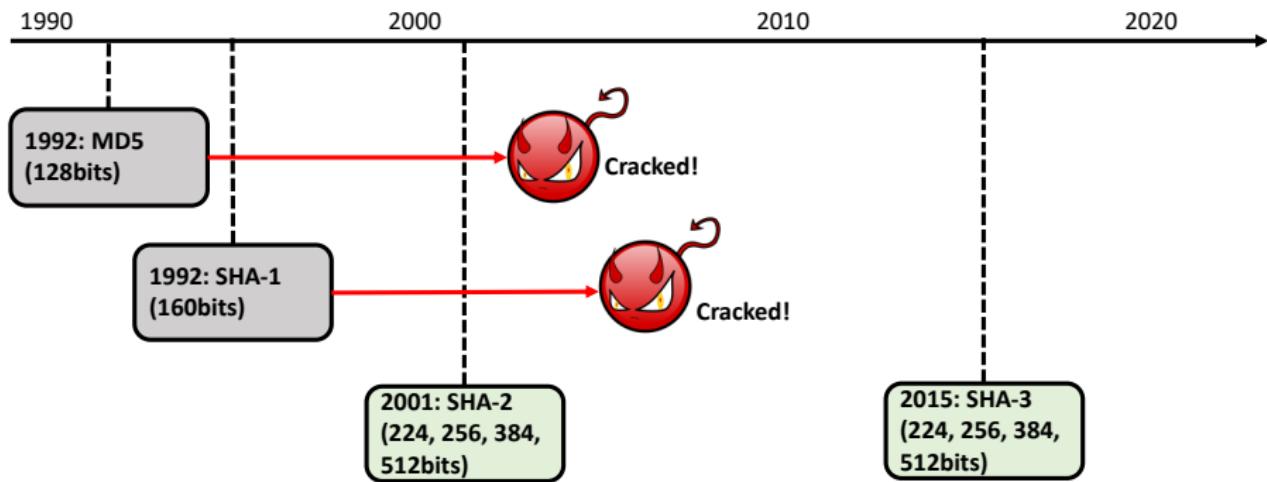
⇒ 別のデータ同士で同じ Hash を得ることが困難なことが必須

Checksum ⊆ Hash と言える。

⁹Hash Table

Hash 関数の種類

MD5, SHA-1, SHA-2 (SHA-256, 384, 512), SHA-3 という Hash 関数がよく知られている。



MD5、SHA-1 は、「同じ Hash(指紋) を生成するデータが割と簡単に見つけられる¹⁰」という致命的な欠陥が発見されている。

¹⁰ 「衝突」と呼ぶ。MD5 の場合は、 2^{20} 程度の計算量でクラック可能。

Hash 関数の選択について

- 理由がなければ SHA-2 シリーズ以降のものを選択する。
 - bit 長は長いほど、衝突するデータが見つけづらい (=強固)
 - ただし、bit 長が長いほど、計算が重くなる
- SHA-1/MD5 は、基本的に互換性の担保のためだけに利用する。但し、Checksum として使う分には問題ない。何が何でも使うな、というわけではない。

IE/Edge こぼれ話

X.509 の公開鍵証明書などはまだ SHA-1 が利用されている場合が多くある。しかし、IE/Edge では互換性の担保を全て無視して SHA-1 のネイティブサポートを全打ち切りしているので、X.509 公開鍵証明書などを JavaScript からネイティブ API を通して扱えない。

JavaScript でデータの Hash を生成してみる。

Hash 生成のコードはこんな感じ。

本人確認の技術

「正しい相手から正しく送信されてきたデータか」？

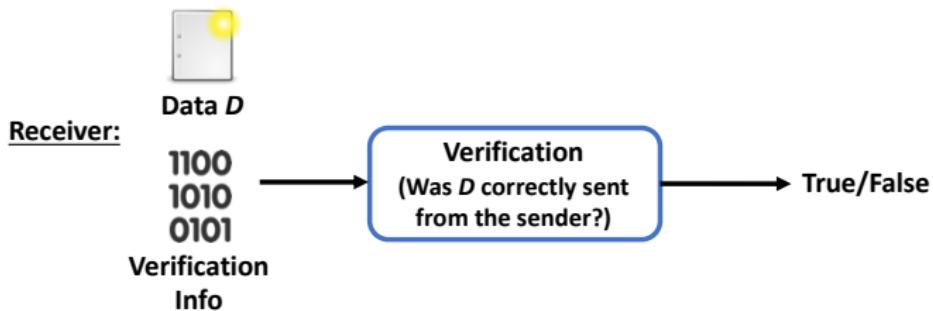
いわゆる「データの真正性と送信元の確認方法」には、大まかに2つの方法がある。

- Message Authentication Code (MAC)
- 署名 (電子署名)

両者とも、送信するデータから MAC/署名という検証用データを生成、元データに付与する形で送信。



受信したデータと、MAC/署名とを突合して、「送信元は意図している相手か？」「データは改ざんされてないか？」を検証。

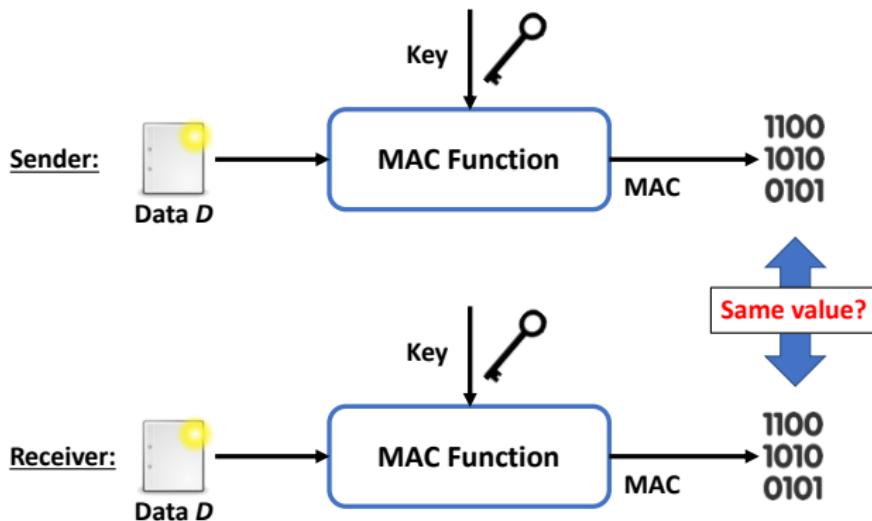


MAC・署名の中身に突っ込む前に、それぞれのざっくりとした定義と pros/cons を説明する。

Message Authentication Code (MAC)

MAC によるデータ真正性と送信者の確認

- 送信側・受信側で共有する鍵を使ってデータ・鍵固有のバイナリ (MAC) を生成する方法。
- 受信側で、送信側と同一の MAC が作れるかどうかをチェック。



MAC の特徴:

- 同じ鍵でも、データが異なれば出力される MAC も異なる。
- 同じデータでも、鍵が異なれば出力される MAC も異なる。

↓

すなわち、受信側で同一の MAC が作れることを確認できれば、

- 鍵を共有する相手から
- 途中の改ざんなしで送られたデータであること

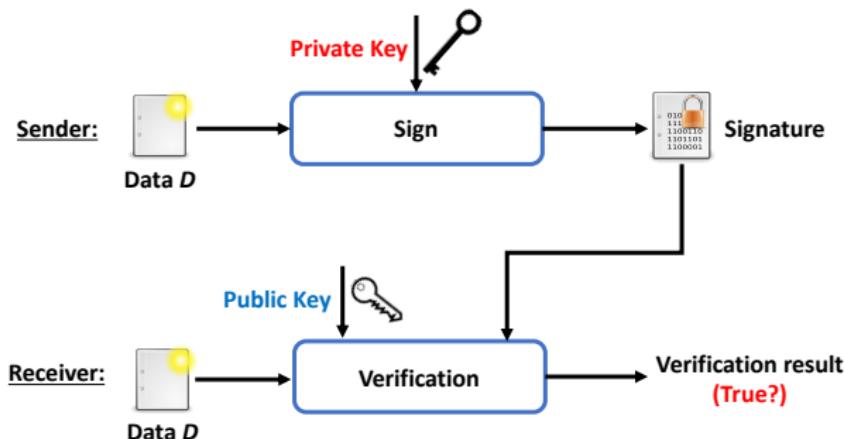
が保証。

署名(電子署名)

公開鍵・秘密鍵ペアをベースとした技術¹¹：

署名によるデータ真正性と送信者の確認

- 送信側は公開鍵・秘密鍵ペアを保有。
- 送信側は、データと自分の秘密鍵から署名を生成。
- 受信側は、受信データ、署名と公開鍵の間の一貫性をチェック。



¹¹ ここでいう公開鍵・秘密鍵ペアは、暗号化に使うものと全く一緒の概念。

署名の特徴:

- データが改ざんされていたら、検証が失敗。
- 意図する相手の秘密鍵¹²で署名が作られていなければ、検証が失敗。
- 検証に使う公開鍵は公開情報(当たり前)
- MACと違って、秘密の情報(=鍵)を事前共有しなくて良い

⇓

すなわち、署名技術は、

- 意図する送信者から
- 途中の改ざんなしで送られたデータなことを
- 事前の秘密情報の共有なしで

保証する。

¹²自分が入手している公開鍵の対となる秘密鍵

MAC と署名の pros/cons

じゃあ署名だけで MAC は不要では？…そういうわけにはいかない。

| | Pros | Cons |
|-----|---|---|
| MAC | <ul style="list-style-type: none">一般的に高速¹³生成する MAC サイズは小さい¹⁴ | <ul style="list-style-type: none">鍵の事前共有が必要 |
| 署名 | <ul style="list-style-type: none">鍵の事前共有が不要 | <ul style="list-style-type: none">一般的に非常に遅い・重い生成する署名サイズは一般的に大きい¹⁵ |

⇒ AES/公開鍵暗号の関係と全く一緒に、使い所を考えて組み合わせて使う、もしくは場合に応じて使い分ける。

¹³ AES (CMAC) とか Hash (HMAC) とかを構成要素としているため。

¹⁴ 通常 128–512bits 程度。

¹⁵ ECDSA は小さく、256–512bits 程度。RSA 系は非常に大きく 2048bits 以上が必要。

共通鍵を使った改ざん検知・本人確認: MAC

Message Authentication Code (MAC) 事始め

Hash-based MAC (HMAC)

HMAC

- 「鍵付き Hash (Keyed Hash)」とも呼ばれる、Hash 関数を利用した MAC 生成方法。
- RFC2104¹⁶ に規定。
- HKDF などの RFC 標準技術や、AWS-Signature-v4 等、各所で利用されている。

「鍵」と「データ」を連結して Hash 関数に入れる、と考えると特徴をイメージしやすい。

¹⁶<https://tools.ietf.org/html/rfc2104>

JavaScript で HMAC を実行してみる

公開鍵を使った改ざん検知・本人確認: 署名

署名 事始め

署名とは？

データそのものに署名を施すのは厳しいので、「データの指紋」に
対してハッシュを施そう。

- RSASSA PKCS1-v1.5
- RSASSA-PSS

RSA なら可能なら PSS を使おう !

JavaScript で RSASSA-PSS を実行してみる

ECDSA

JavaScript で ECDSA を実行してみる

運用 Tips

署名検証のブートストラップの問題

署名の検証用の公開鍵が正しいことはどうやって保証するの？

⇒ App に固定インストールか、Verisign あたりに署名してもらう必要…

⇒ PKI に頼って検証用の公開鍵の信頼性を担保するしか、現状は方法がない。

署名・MACの使い分け

処理の重さで使い分けると幸せになれる。

- 署名は手続きのイニシエーションに使う
- MACは、なんども繰り返すような本人確認に使う

例えば。。。

- 1 署名を付与して、ECDH-ephemeral の公開鍵を交換。
- 2 ECDH-ephemeral + AES で MAC の鍵を共有。
- 3 以降の大規模データのやり取りは MAC で本人確認を実施。

など。

まとめ

まとめ

お疲れ様でした。



次回は

宣伝: iTransfy by Zettant

簡単・安全にファイル転送ができる

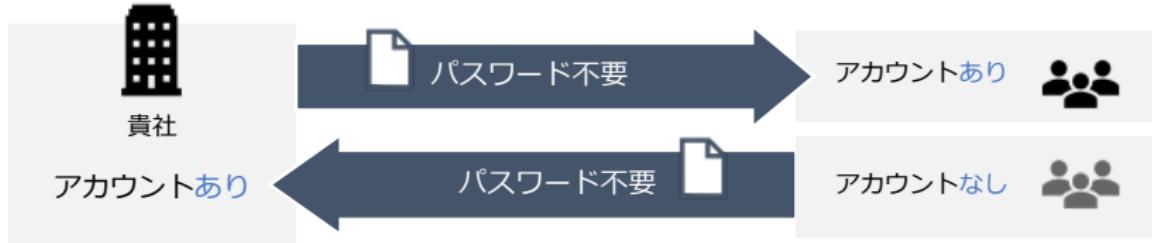


iTransfy for biz

<https://www.itransfy.com>

アカウント登録で、パスワード入力の手間が省けます

クライアント/協力会社等へファイルを送りたい、また送付してほしい時の手間を軽減



宣伝: 株式会社ゼタント



ゼタントはのミッションは、
「自分の身は自分で守ることができる世の中にする」
ことです。
共感してくれる仲間を募集しています！

問合せ先: recruit@zettant.com
会社 URL: <https://www.zettant.com>