

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 24 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。

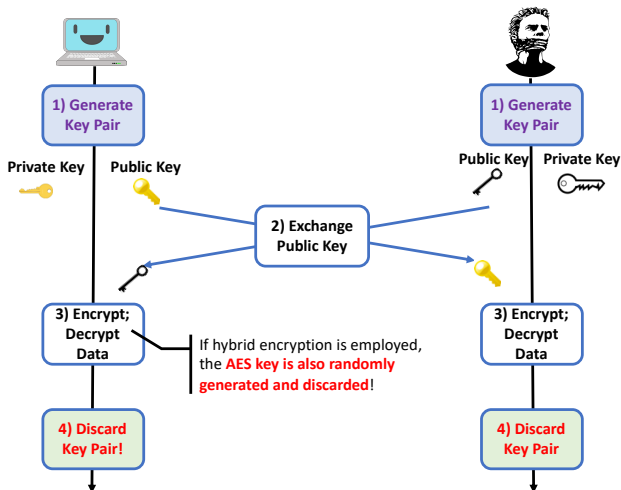
今回は、第 3 回の最後に懸案事項だった「データのやり取りしてる相手って本当に正しい相手？」を保証する方法を学んでいく。

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 Secrecy**¹を担保する運用方法。

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

まずはじめに、「送られてきた 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>)

⁴RSA-PSS (PKCS#1 RFC8017 <https://tools.ietf.org/html/rfc8017>), ECDSA (NIST 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います。
- 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 が利用可能なこと

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

細かく Hash/MAC/署名の説明を聞きつつ、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。

デモの絵

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・前々回使った「リモートサーバに登録する」というところは、簡略化のため省略した。興味があれば、前回のコードを公開鍵暗号に拡張して、ネットワークを介して E2E セキュリティしてみよう！

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

■ 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
```

■ 依存パッケージのインストール

```
$ yarn install
```

■ 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

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のが必須

■ Hash

- データのエラー・改ざん検知
- 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

目的としては、Checksum \subseteq Hash と言える。

⁹Hash Table

Hash の種類

MD5/SHA1/SHA256 から SHA3 へ

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

MAC と署名とは？

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)

「鍵付きのハッシュ (Keyed Hash)」だと思えば良い。

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

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

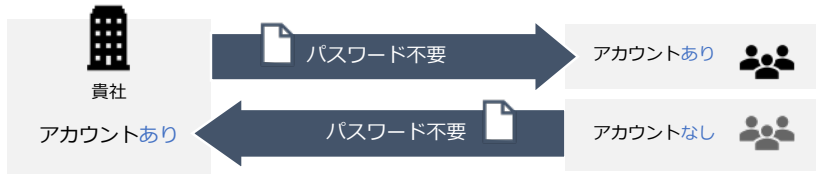


iTransfy for biz

<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>