

# JavaScript による **End-to-End** セキュリティ

## 第 3 回 公開鍵暗号はどうやって使えばいいのか？ 編

栗原 淳

2019 年 10 月 17 日

はじめに

# はじめに

第1回と第2回では

- End-to-End (E2E) セキュリティの原則と必要性
- Web サイトでの E2E セキュリティ実践のため、JavaScript で暗号 (AES) を正しく・安全に利用する方法

を勉強した。

ところで、AES(共通鍵暗号)とは別に、「公開鍵暗号」というのが存在する。

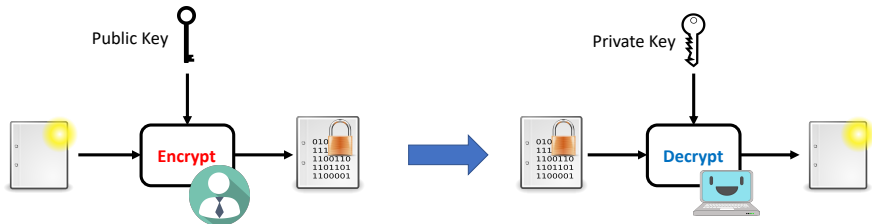
# 公開鍵暗号って？

既知だと思うが、まずざっと定義しておく。

## 定義: 公開鍵暗号

以下のステップで暗号化・復号が行われる暗号方式のこと

- 1 特殊な数学的条件を満たす鍵ペア「公開鍵  $PK$  と 秘密鍵  $SK$ 」を生成
- 2  $PK$  は公開、 $SK$  は秘匿
- 3 データ  $D$  を  $PK$  によって暗号化して、暗号化データ  $X$  を生成
- 4  $X$  は  $SK$  によってデータ  $D$  に復号される。



Anyone can encrypt information only with public key without knowing private key!

Only the private key owner can decrypt the information encrypted under its paired public key.



暗号化・復号の鍵を分けて、暗号化の鍵を公開してしまうことでパスワードなどの共有が不要になる。

⇒ AES などにはない、非常に強力な暗号化の概念。

今回は正しく・安全に公開鍵暗号を使っていくためのお話。

#### この講義で最終的に学びたいこと

- 公開鍵暗号はどういうものか。AES と比べた pros/cons。
- RSA 暗号と楕円曲線暗号<sup>1</sup>の違い。
- AES と公開鍵暗号を組み合わせデータ暗号化するために。

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。

---

<sup>1</sup>楕円曲線 Diffie-Hellman を取り上げる

# 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:

- Git が使えるようになっていること
- Node.js 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

# 公開鍵暗号の使い方 事始め



# 公開鍵暗号の種類

公開鍵暗号の定義「特殊な数学的条件を満たす鍵ペアを生成」



この「数学的条件」に複数の種類が存在。

JavaScriptに限らず、各種環境で利用可能な代表的な公開鍵暗号：

- 素因数分解に関する条件  
→ **RSA 暗号**
- 楕円曲線上の離散対数に関する条件  
→ **楕円曲線暗号 (Elliptic Curve Diffie-Hellman; ECDH)**

この2つの使い方、注意ポイントを今回は取り上げる。

## RSA Cryptography

言わずもがな、公開鍵暗号の代表的な手法

- 1977 年、Rivest-Shamir-Adleman の 3 名により発明。2000 年に特許期間満了 (現在特許フリー)。暗号化以外に「署名」の手法への応用も有名。
- RFC 8017 (PKCS#1 v2.2)、ANSI X9.31、IEEE 1363、CRYPTREC 等、各所で標準に採用。
- 鍵長は 1024–4096bits が標準的に使われている。<sup>2</sup>
- 暗号化・署名の際には、元のデータにパディングが必要。パディング方法によりセキュリティが大きく左右される。<sup>3</sup>

<sup>2</sup>原理的には無限に伸ばせる。

<sup>3</sup>RSA-OAEP(暗号化)、RSA-PSS(署名) が現状ベターな方法。これを話す。

# 楕円曲線暗号のさわり

## Elliptic-Curve Cryptography

楕円曲線という数の世界での「離散対数」を使った方式の総称<sup>4</sup>

- 1985 年頃、Victor Miller、Neal Koblitz により独立に考案。
- TLS (RFC8442) 等で標準化。Bitcoin など多方面で利用。
- Diffie-Hellman(DH) を楕円曲線上で実行するのが ECDH、DSA を楕円曲線上で実行するのが ECDSA。
- 鍵長は、256–521bits が標準的に使われている。
- ECDH は「ECDH-Ephemeral」という方法で実行することで、普通に使うより**安全性が大きく向上**する。<sup>5</sup>

<sup>4</sup>普通に離散対数問題を使うより、楕円曲線上でやることで安全性を担保する鍵長が短くなる。

<sup>5</sup>Forward Secrecy(第 1 回のスライド参照) を担保する。

# AES と比べた公開鍵暗号の Pros/Cons

	Pros	Cons
AES	<ul style="list-style-type: none"><li>・ 安全性を担保する鍵長が短い (128bits～)</li><li>・ 一般的に <b>高速</b>・ SoC での最適化も望める<sup>6</sup></li></ul>	<ul style="list-style-type: none"><li>・ パスワードなどの <b>事前共有が必要</b></li></ul>
公開鍵暗号	<ul style="list-style-type: none"><li>・ パスワードなどの秘密情報の <b>事前共有が不要</b></li></ul>	<ul style="list-style-type: none"><li>・ 安全性を担保する鍵長が長い (RSA: 2048bits～)</li><li>・ 一般的に <b>非常に遅い・重い</b></li></ul>

⇒ 使い所を考えて組み合わせて使うことがオススメ (後述)

<sup>6</sup>Intel AES-NI

# 安全性を担保する鍵長が大きく違うのはどういうこと？

AES と比べた RSA ・ 楕円曲線暗号の鍵のビット長比較<sup>7</sup>。横 1 行  
がだいたい同じくらいの安全性と言われる。

AES	RSA	楕円曲線
128	3072	256–383
192	7680	384–511
256	15360	512–

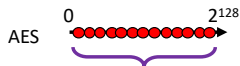
AES に比べて、楕円曲線で倍、RSA に至っては 24 倍以上の鍵長  
を使わないと、同じくらいの安全性を担保できない。

※ 鍵長は長ければ長いほど処理がどんどん重く・遅くなって  
いく…

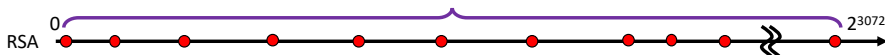
<sup>7</sup>Recommendation for Key Management, Special Publication 800-57 Part 1 Rev. 4, NIST,  
01/2016. <https://csrc.nist.gov/publications/detail/sp/800-57-part-1/rev-4/final>

「AES-128 が、RSA-3072 と同じくらい」というイメージは、以下のように説明できる。

- AES: 数値 =  $0, 1, \dots, 2^{128} - 1$  のうち、どれか 1 つが鍵。
- RSA: 特殊な条件を満たす数 = 素数 2 個の積 (合成数) を選んで、公開・秘密鍵を求める。



Equivalent number of key candidates



総当たりした時に「当たる」確率を揃えるには、RSA はその分巨大な数まで候補にしないとならない。

# RSA 暗号を使ってみよう

# RSA 暗号って何？



# RSA 暗号を使うためのお作法

Padding:

RSA OAEP <https://tools.ietf.org/html/rfc8017>

PKCS1-v1.5 padding はやばい。Cryptrec から落ちた。

<https://www.cryptrec.go.jp/method.html>

<https://tools.ietf.org/html/rfc8017> 既知の攻撃が知られており、

RSASSA, RSAES 共に PKCF1-v1.5 は非推奨@RFC

でも Node.js では OAEP 非サポートなので作った

# RSA-OAEP によるデータ暗号化を試みよう

楕円曲線暗号 (ECDH) を使ってみよう ECDH(楕円曲線 DH) は、公開鍵暗号というより「匿名での鍵共有方法」というイメージ。

# ECDHって何？

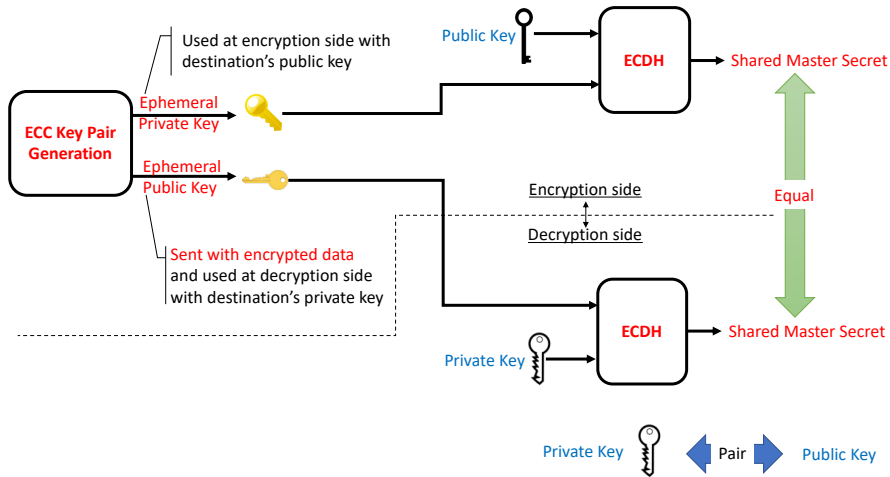
TLS <https://tools.ietf.org/html/rfc8422>

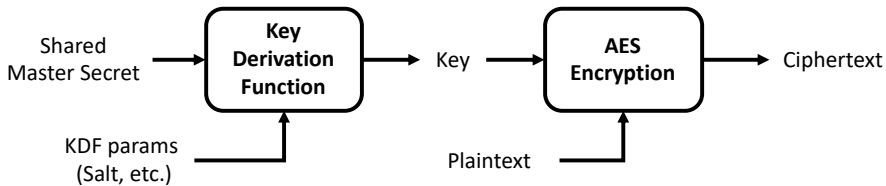
JOSE だと Concat KDF を使うだけ。

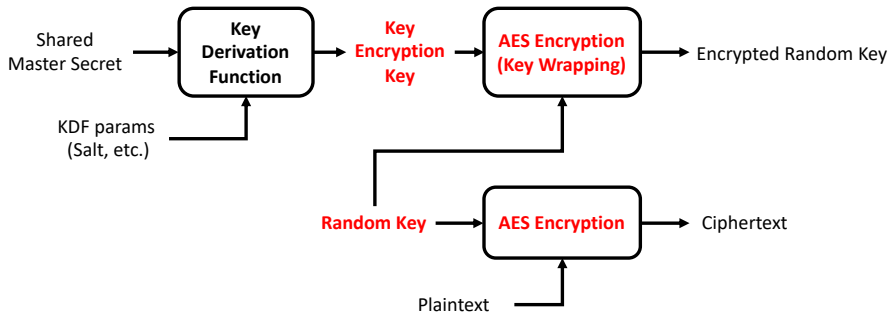
<https://tools.ietf.org/html/rfc8037>

直接 Concat KDF を暗号化の鍵にするか、あるいは Concat KDF の結果を AESKW の鍵として Content Encryption Key を暗号化するのに使う。

# ECDH Ephemeral (ECDHE)







# ECDH によるデータ暗号化を試みよう

今回は HKDF で暗号化してみる。



# AES と公開鍵暗号のいいとこ取り

# 公開鍵暗号と **AES** の比較

# ハイブリッド暗号化

AES の暗号化データをガンガン使い回せる！  
msgpack-light を使ったコードを提供



# まとめ

# まとめ

お疲れ様でした。

- 公開鍵暗号を利用する際のお作法を学んだ。