JavaScript による End-to-End セキュリティ

AES はどうやって使えばいいのか？ (共通鍵暗号の使い方) 編

栗原 淳

August 23, 2019

はじめに

はじめに

前回 (第 1 回) 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 での暗号 (AES) の利用のさわり

を勉強した。

E2E セキュリティの重要性はわかった。
AES を使ってみることもできた。

でも、実際の App で正しく・安全に AES を使うにはどうすべきなのか？

今回は正しく・安全にAES を使ってみる方法、についてのお話。

この講義で最終的に学びたいこと

- パスワードを使って AES 暗号化はどうすればいいか？[1]
- 固定バイナリ値を使って AES 暗号化はどうすればいいか？[2]

たったこれだけ。

[1] RFC8018 PBES2 https://tools.ietf.org/html/rfc8018 による AES 暗号化

[2] RFC5869 HKDF https://tools.ietf.org/html/rfc5869 による鍵導出と AES 暗号化

たったこれだけでも、気をつけなければならない「重要なお作法」がある。

お作法を守る・守らないで安全性は大違いなので、注意しなければならない。[3]

[3]世の中のソフトウェア、全くお作法を守ってないのがあって…危険…最近だと php の `hash_hkdf()` がお作法守ってなかった (2018 年)。

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

AESの使い方 事始め

AES を使う際に気をつけるお作法は、ざっと 3 点。

1. AES で使う鍵のランダム具合
2. AES で使う鍵を総当りする際の大変さ [4]
3. AES の利用モードの安全性

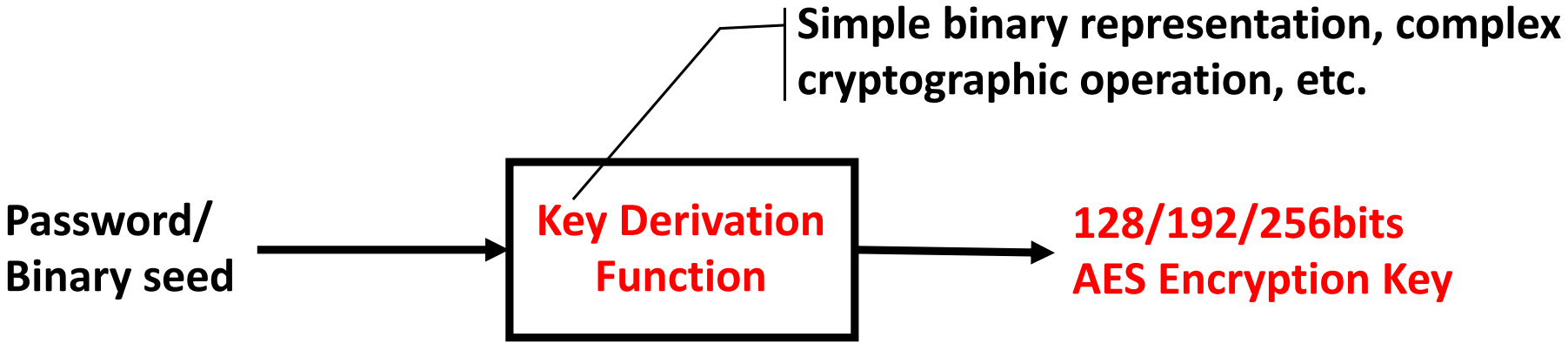
つまりどういうこと？

[4] 1 点目と 2 点目は似ているようで異なる。

準備: パスワードとかを使った AES 暗号化のポイント

パスワード ≠ AES 暗号化の鍵

パスワードやバイナリ値を元にして AES 暗号化するためには、
「パスワード等を変換し、AES 暗号化の鍵を導出」することが必要



1: AES で使う鍵のランダム具合？

⇒ 過去の利用履歴も含めたランダムさのこと

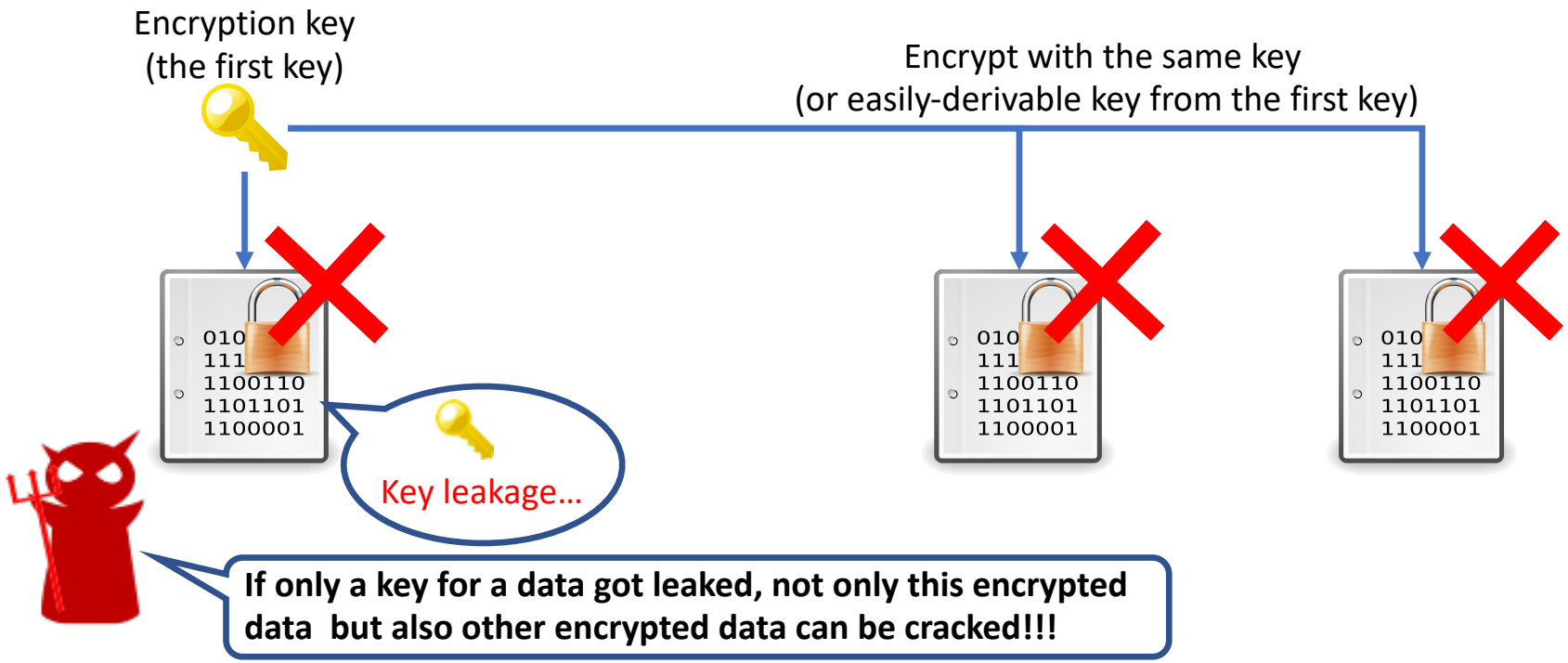
つまり…

- 過去に暗号化に使った鍵は二度と使わない
- 暗号化の鍵は、過去の鍵から[5]は容易に導出できないものへと毎回ランダム変更する

ということ。

[5]および未来に使う鍵からも

…なぜか？
⇒ 鍵が１つ漏れてしまうと、過去の暗号化データまで一網打尽…。

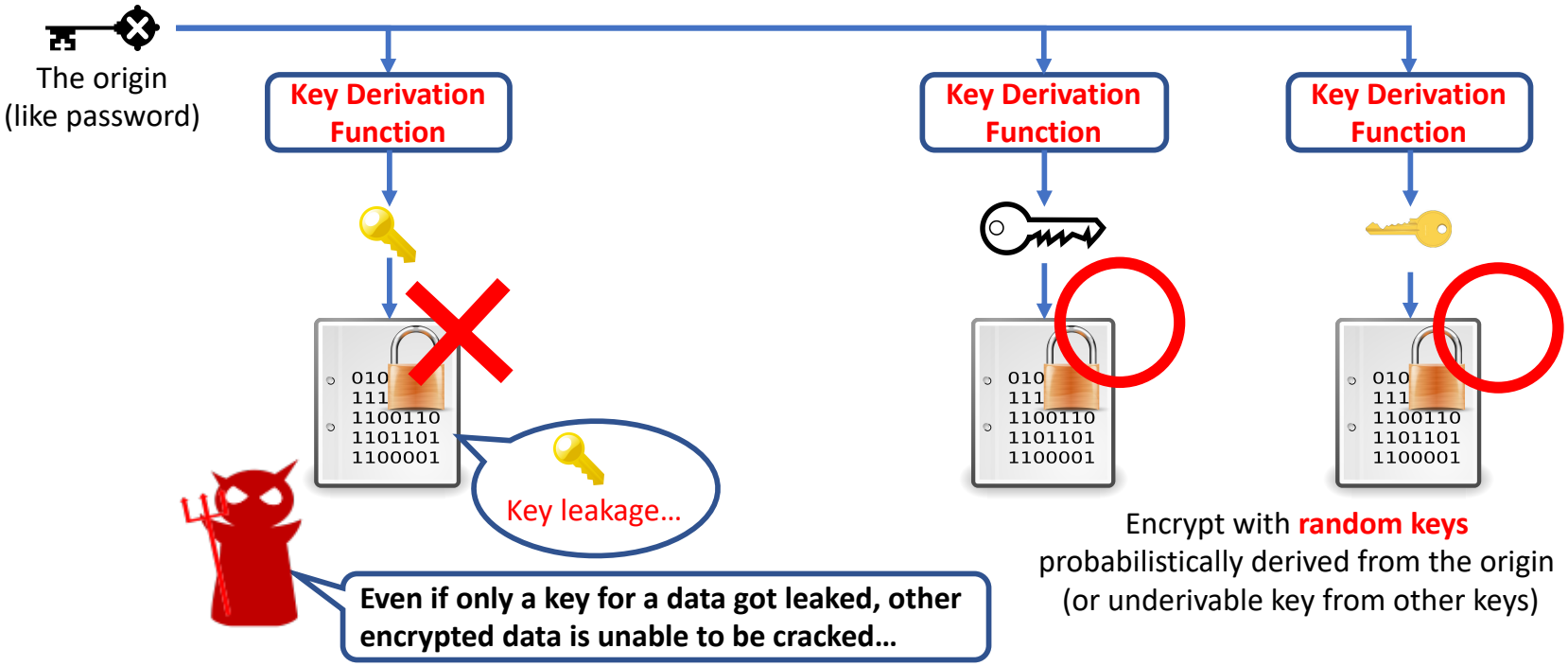


なので、万一鍵が１つ漏れちゃったとしても、他の暗号化データにまで影響が出ないことを保証しなきゃならない。[6]

[6] これを保証することを (Perfect) Forward Secrecy とか呼ぶ。

だが、暗号化毎のパスワード等のランダム変更は非現実的。

⇒ 固定パスワード等からランダムに鍵を導出する方法を使う[7]。



※ただし、固定パスワード等そのものが漏洩した場合はこの場合でもアウトなことに注意

[7] PBKDF2 (RFC8018), HKDF (RFC5869)

2: AESで使う鍵を総当りする際の大変さ？

⇒ 総当たり攻撃のためのコストのこと。
※特にパスワードを使って暗号化する場合に重要

暗号化データに対する総当たり攻撃

鍵の候補を全通りを一覧で用意して、「当たり」を見つけるまでとにかく復号を繰り返すこと。

つまり総当たり攻撃のコストは、「ストレージ量」と「計算量」。
このコストを払うことが非現実的に高くなければヤバい。

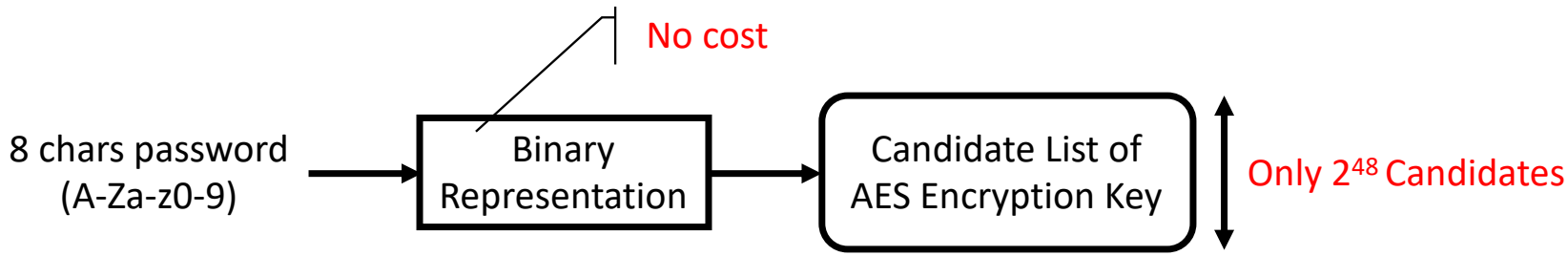
8 桁パスワードを単純にバイナリ化して鍵としてしまうと…

大小英数字 8 桁パスワードは $62^8 < 2^{48}$ 通り。

⇒ 48bits の全通りの準備は、高々1.5PB。

⇒ ストレージなしでも、パスワード候補を都度バイナリ化するだけで復号を試行可能。

割と簡単に「当たり＝バイナリ鍵」が見つかってしまう。[8]



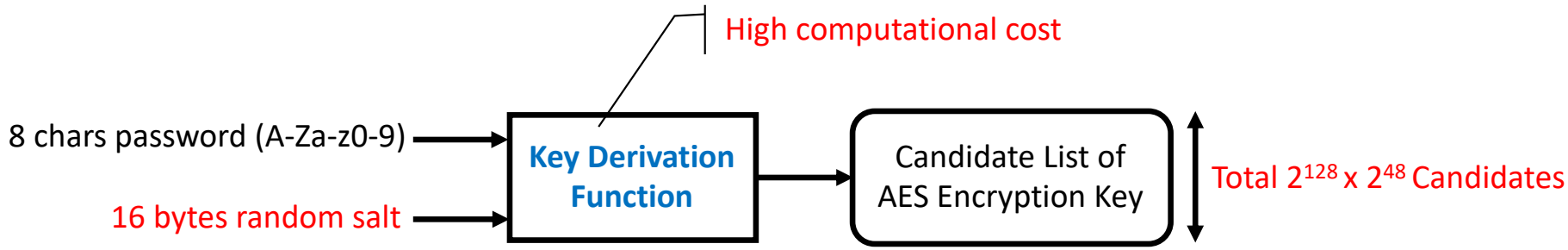
[8] 2009 年当時でもスパコンを使って 60 時間とか。今だと GPU で並列化すればもっと高速になる。https://web.archive.org/web/20180412051235/http://www.lockdown.co.uk/?pg=combi&s=articles

なので、短いパスワード等から鍵を作るときは、コストが膨大になるような変換をする。

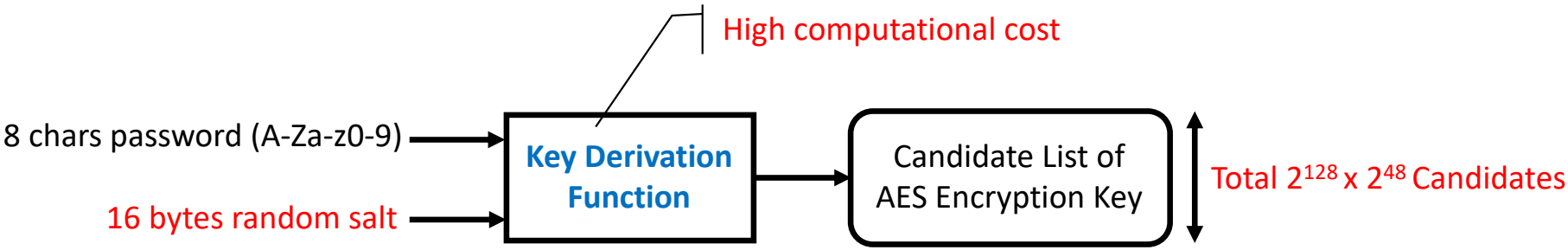
パスワード等から暗号化の鍵を作るとき、

- 毎回使い捨てのランダム値 (Salt と呼ぶ) と混合して、AES 暗号化の鍵のランダム性を上げる。
- 計算コストの高い演算を使う。

という処理を行う。[9]



[9] PBKDF2



- ランダムな Salt と混合することで、鍵候補全通りの事前準備のストレージが膨大になる
- ストレージなしで試行しても、計算コストの高い演算のせいで、鍵候補を都度生成→復号の計算コストが莫大になる

「お作法 1」と合わせて 1 つの関数で実行することが多いが、AES 暗号化の鍵を作る際に意識する重要なポイント。

3: AES の利用モードの安全性？

⇒ AES の API で設定できる利用モード ('AES256-CBC' とか) と、そのパラメタ設定の適切な設定が必要。

AES の「利用モード」

AES の処理 1 回で暗号化できるのはたった 16bytes にすぎない。長いデータを連続で暗号化するために、暗号化処理を連続して組み合わせる方法が利用モード。

「とりあえず AES を使う」ためのポイントは 2 つ

- 初期ベクトル (IV) というパラメタは都度ランダム値にする [10]。
- CTR モード・CBC モードあたりを使う。ECB モードは絶対に使わない。

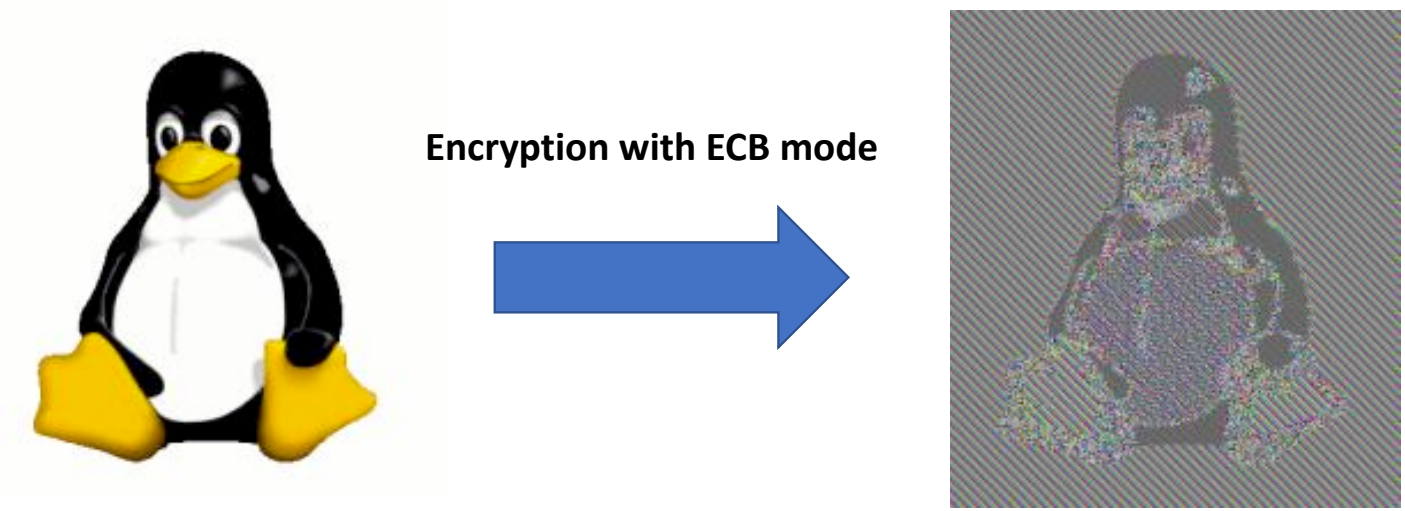
前者、「過去に暗号化したデータとの相関をなくす」ために必要なパラメタ設定。
後者、ECB モードは論外 (これが言いたいこと)。

[10] API によって、ナンス (Nonce) というパラメタもあればそれも。

どうして ECB モードは論外なのか？

⇒ 元のデータの中で「同じ値のブロック[11]」は、暗号化データにおいても必ず「同じ値のブロック」になる。

⇒ 暗号化されてても中のデータが何かというのが予測可能…



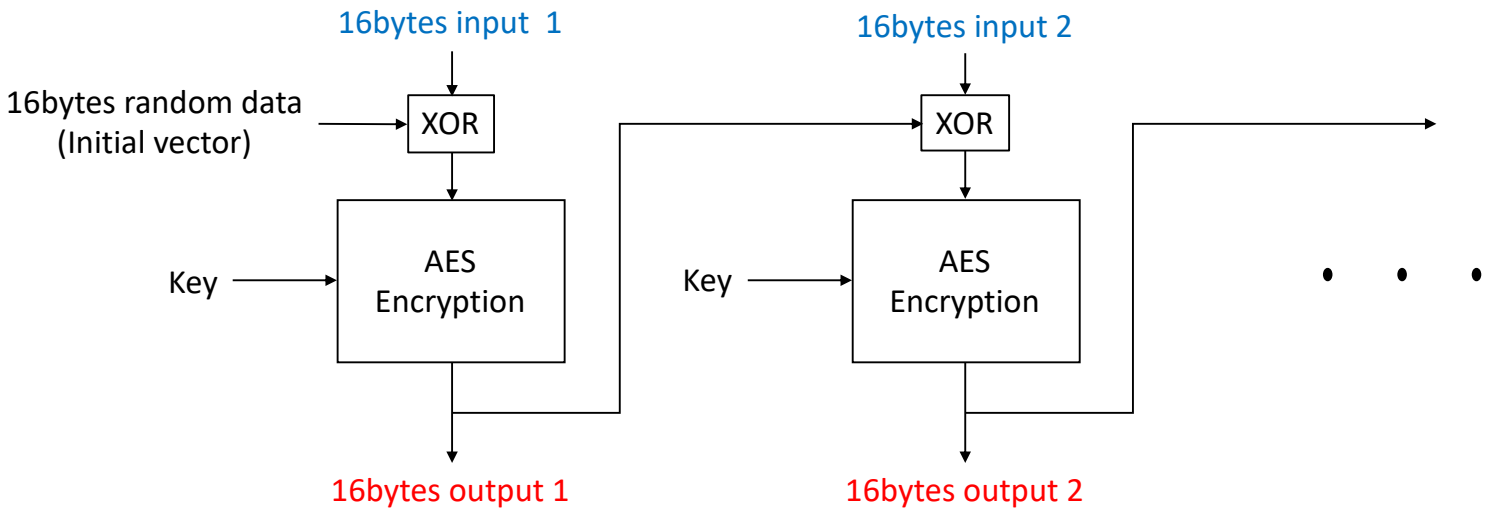
Original images are given by Larry Ewing (lewing@isc.tamu.edu)

というわけで、JavaScript 以外でも、たとえ選べたとしても絶対に ECB モードは利用してはいけない。

[11] 1 ブロックは 16Bytes 単位

ECB モードと違って、CBC モードではそういうことが起きない。

- 先頭の 16Bytes はランダムな初期化ベクトルと混ぜる
- 前の 16Bytes のデータを継承して次の 16Bytes を処理



CBC モードの 16Bytes 毎の処理

AESの使い方: とりあえず暗号化してみよう

パスワードで暗号化してみる

pbkdf 使え

バイナリ鍵で暗号化してみる

hkdf 使え

危ない暗号化モードで暗号化してみる

ecb とか論外だから cbc とか使え

AESの使い方: 細かめの解説

PBKDF2 の使い方 in JavaScript

jscu なら動くよ

HKDF の使い方 in JavaScript

暗号化モードの設定

cbc 云々。
js でのサポート具合を列挙

まとめ

まとめ

お疲れ様でした。

-

次回以降…リクエスト次第ですが、

- 公開鍵暗号とその使い方
- 「情報が改ざんされてない」ことを保証するために（電子署名と MAC）
- RFC とアルゴリズム・フォーマット

などを予定。

宣伝 1

E2E 暗号化ファイル転送サービス「iTransfy」を提供しています。

宣伝 2

Zettant ではイケイケの仲間を募集しています。

1. 今回は共通鍵暗号
2. 公開鍵暗号& Hybrid Encryption
3. ハッシュ・署名と HMAC
4. 超マニアック講座：RFC とアルゴリズム・フォーマット

Appendix

This page is not counted.