JavaScript による End-to-End セキュリティ

入門編

栗原 淳

August 1, 2019

はじめに

はじめに

この講義では

- End-to-End (E2E) セキュリティの原則
- Web サイトでの E2E セキュリティ実践のため、JavaScript での実装方法
  - ブラウザ側
  - サーバ側 (Node.js)

のさわりを学ぶ。

モダン Web サイトと End-to-End セキュリティ

Web サイトにおける昨今の情勢

- EU における General Data Protection Regulation (GDPR) の施行 (2018 年)
- GDPR に続いて、カリフォルニア、南米、オセアニアで類似の法律の制定の動き
- 日本においても、2020 年に個人情報保護法の改正法案提出の見通し

⇓

企業にとって、「正しく」「強固」に
ユーザデータ、ユーザプライバシを保護することは必須の事項

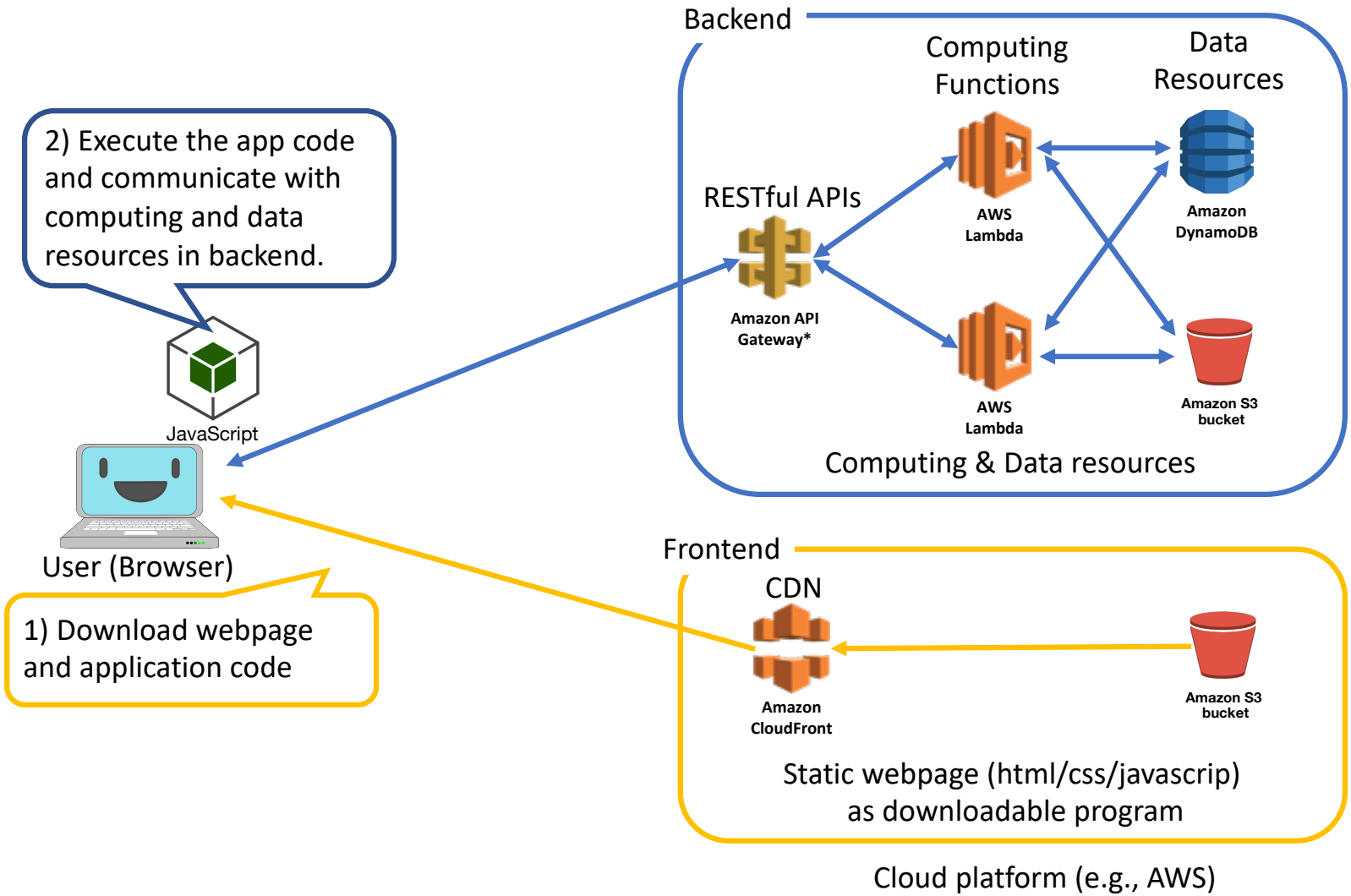
最近流行りの **Web** システム

- クラウドプラットフォーム上で構築
- 「サーバ」のない (サーバレス) 構成
- JavaScript (ReactJS など) を多用した、Single Page Application 構成

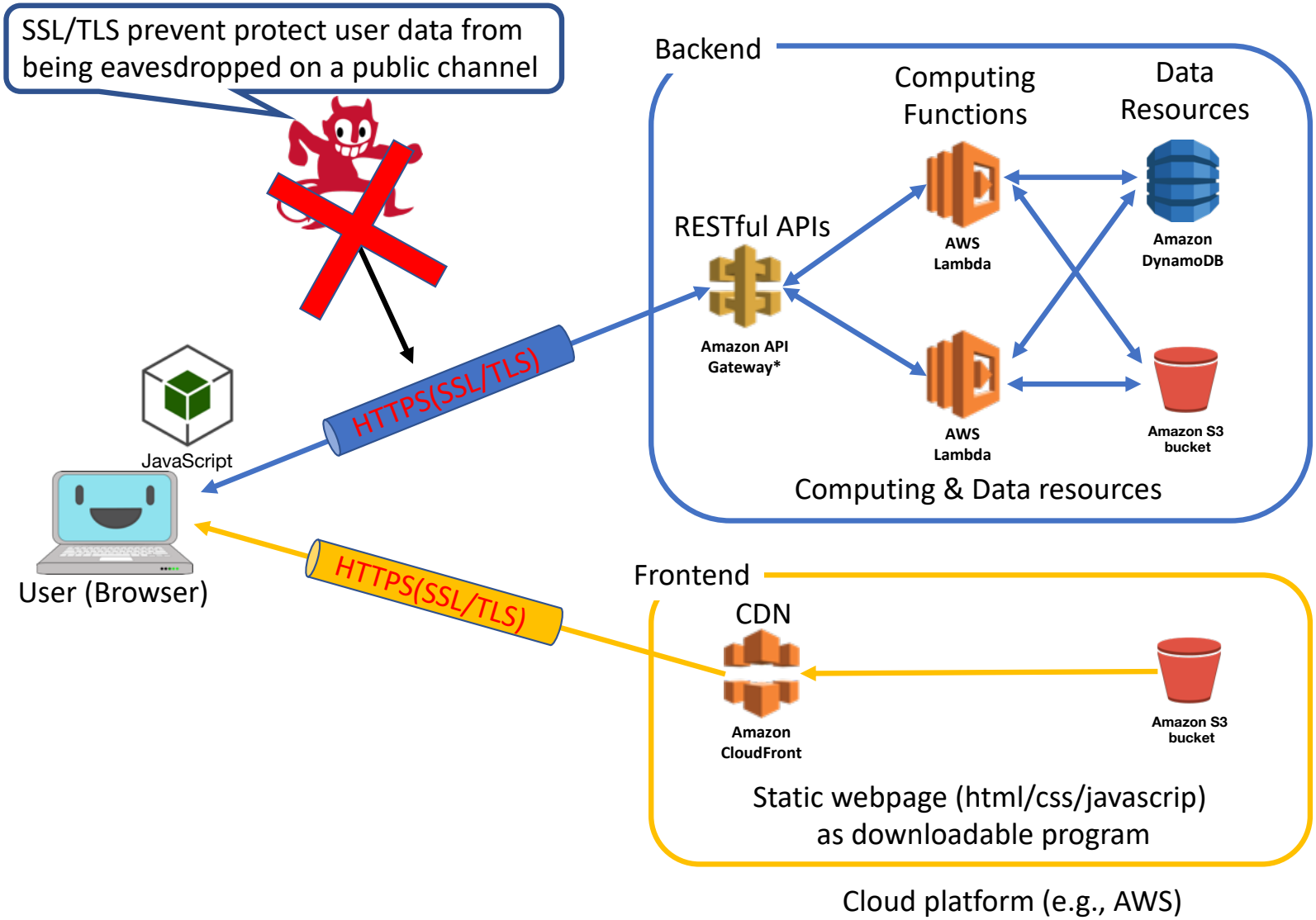
⇓

ユーザの手元で計算を実行する機会の増加

AWS を例にした典型的な構成:



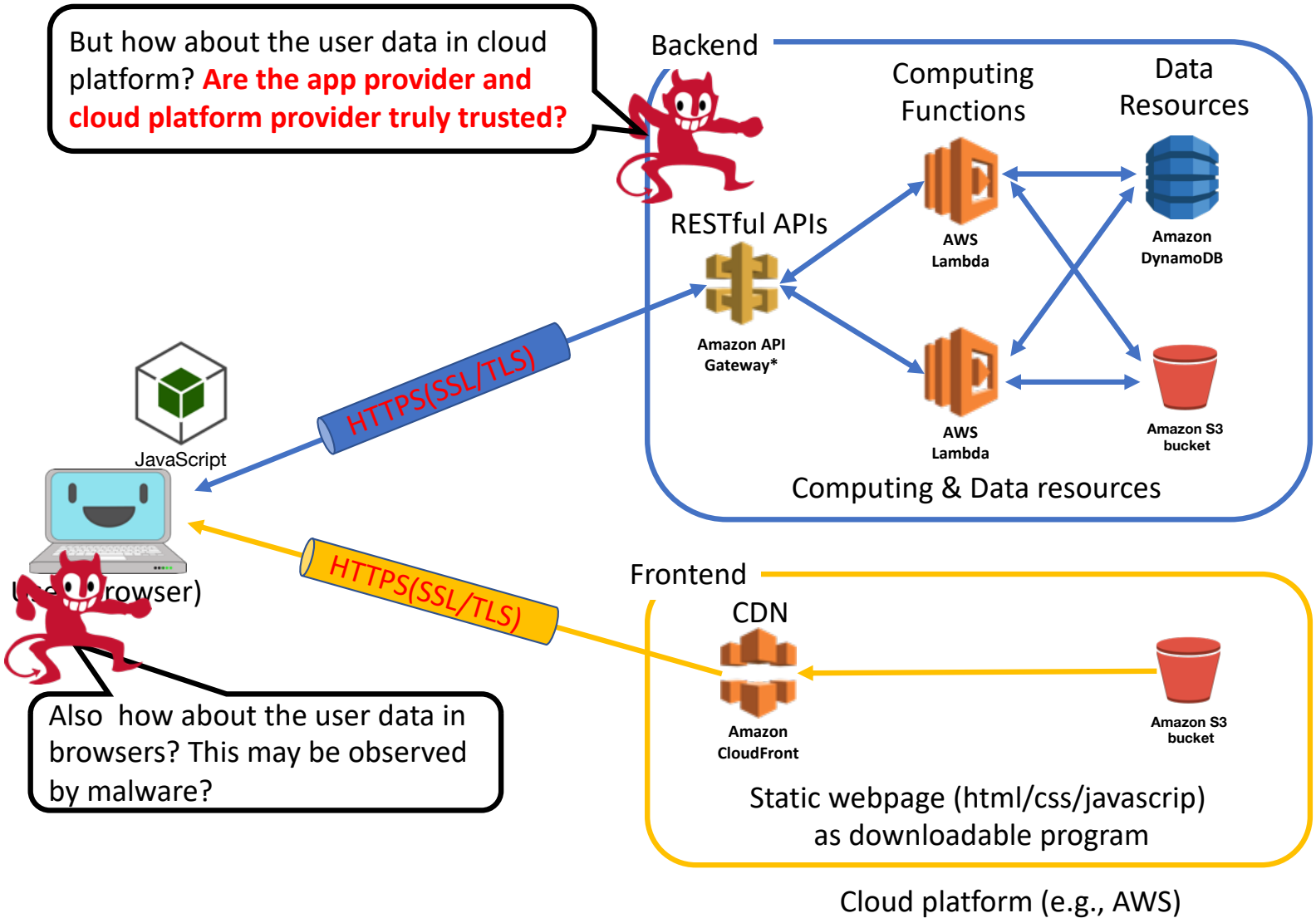
AWS を例にした典型的な構成:



通常、ユーザ・クラウド間の HTTP 通信路は SSL/TLS で保護

⇒ HTTP 通信路から外部の盗聴者へのユーザデータ漏洩を防止

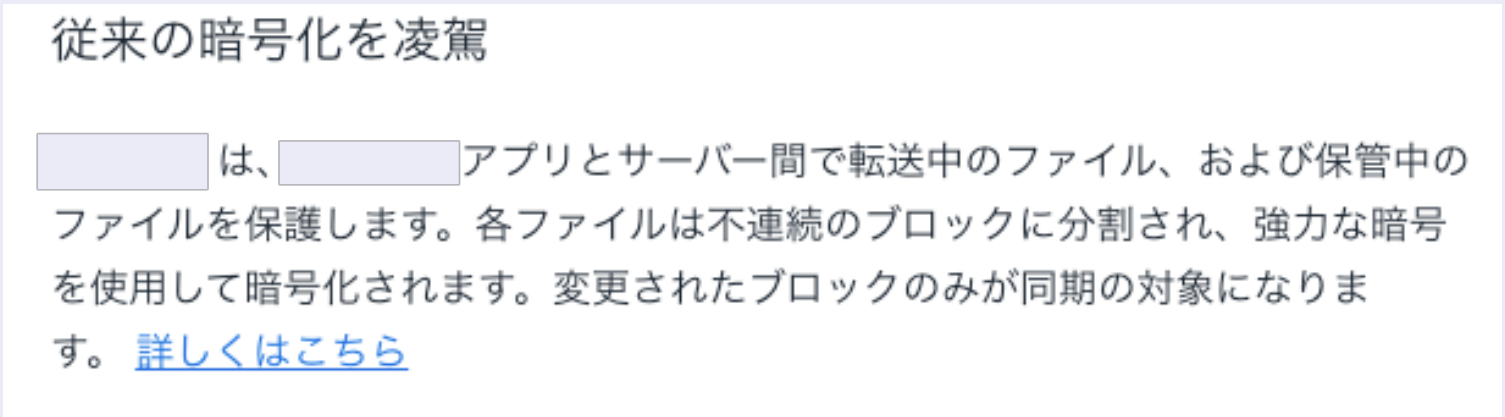
AWS を例にした典型的な構成:



しかし、クラウド PF 内・ブラウザ内のデータ保護は……？

「暗号化しているから安全です」という叙述トリック

(Web とはちょっと違いますが…) 某クラウドストレージ事業者の例



クラウドベースの (Web) サービスでよくある文言：

- (SSL/TLS で) 転送中のデータを暗号化して保護
- ストレージに保存されるデータは暗号化して保護

- (SSL/TLS で) 転送中のデータを暗号化して保護
  ⇒ 公開通信路の盗聴からデータを保護
- ストレージに保存されるデータを暗号化して保護
  ⇒ ストレージ自体が盗まれた時や、第三者のストレージを使っている場合のデータ漏洩を防止

いずれも事業者に対しての秘匿性を担保しているわけではない[1]

⇓

(望む・望まないにしろ) 事業者はユーザデータを不必要に取得

[1] 事業者はデータを見放題ということ。

このようなクラウドサービス・Web App を作ることは：

- ユーザにとって：共有不要な相手とデータを共有している
- 事業者にとって：昨今のプライバシ・セキュリティ要求の高まりから、無用なリスクを背負いこむ可能性が大

今後、Web App を作っていくにあたって

「必要な相手とだけ」確実に・正しく、データを共有できるように、適切なデータ秘匿が必要

データの秘匿性・プライバシを謳うサービス

- Tresorit[2]:
  事業者・サーバに情報が漏れないことを謳ったクラウドストレージサービス。Dropbox に近い。
- KeyBase[3]:
  事業者・サーバに情報を漏らさず、メッセージ・ファイル共有（クラウドストレージ）が可能な SNS。
- Signal[4]:
  事業者・サーバに情報を漏らさないメッセージング・通話アプリケーション。「最も安全な」チャットサービスと呼ばれており、各類似サービス (WhatsApp など) にプロトコルを提供。

北米・EU 共に、スノーデンの事件以降、事業者にも情報を与えない End-to-End 暗号化を謳ったサービスが強く注目を浴びている。

[2] https://tresorit.com/
[3] https://keybase.io/
[4] https://signal.org/

End-to-End セキュリティとは

End-to-End (E2E) Principle[5]

The end-to-end principle is a network design method in which application-specific features are kept at communication end points.

(アプリケーションの機能はネットワークシステムの終端で実装されるべきという原則)

[5] J. H. Saltzer et al., “End-to-End Arguments in System Design”, in Proc. ICDCS 1981, pp. 509–512, Apr. 1981.

アプリケーションのセキュリティについての E2E Principle:

End-to-End (E2E) セキュリティ

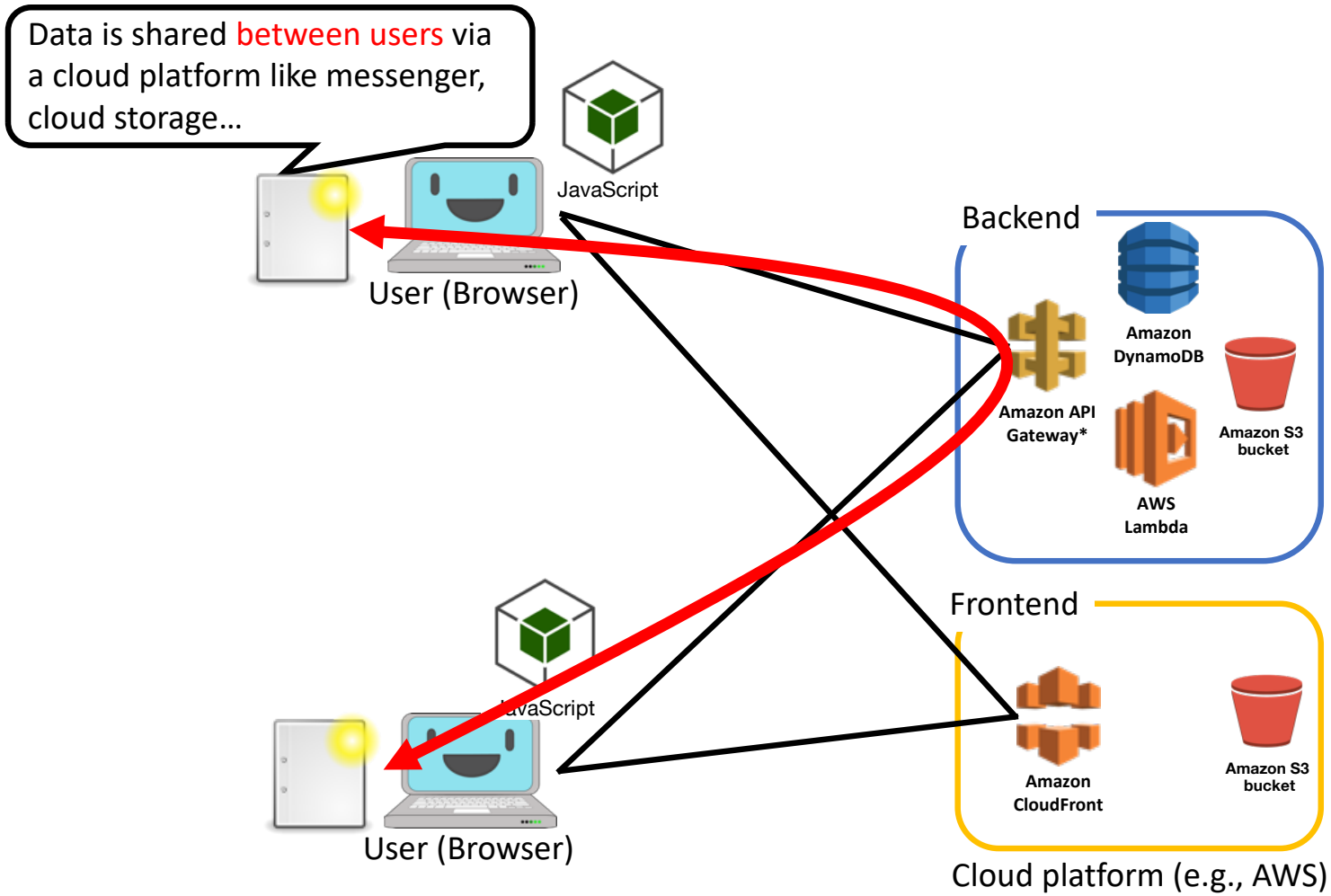
情報を共有する主体同士 (End-to-End) について、セキュリティ3原則を担保する。

- 情報の秘匿性 (主体のみで情報を共有可能) ⇒ E2E 暗号化
- 主体・情報の真正性 (主体が生成した情報であることを証明)
- 情報の可用性 (主体同士が正しく情報を利用可能)

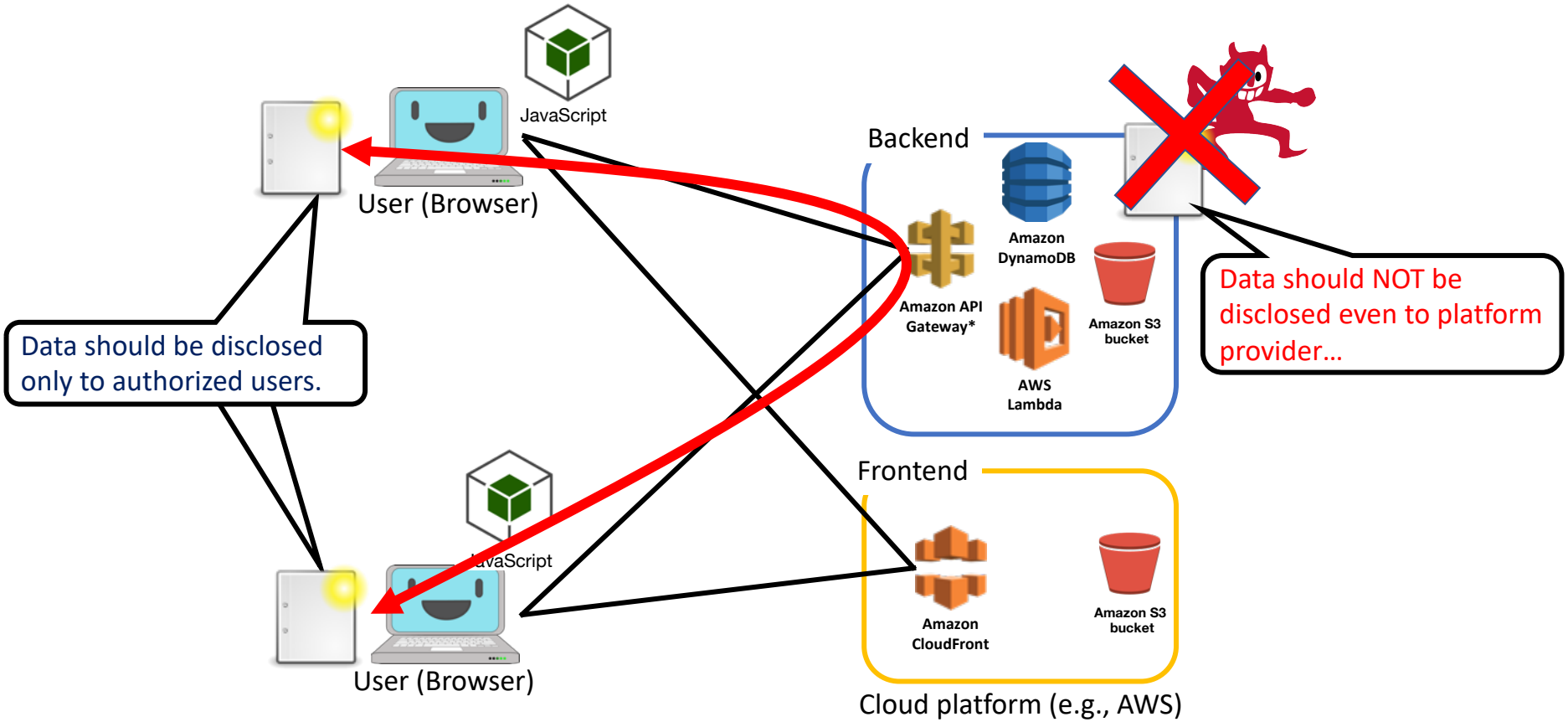
ポイント：アプリケーションにおいて「情報を共有する主体」は一体何か。

- アプリケーションを利用するユーザ同士？
- サーバ・クライアントアプリ同士？
- 他？

例: クラウドプラットフォームを介し、ユーザ同士が情報を共有する主体の場合

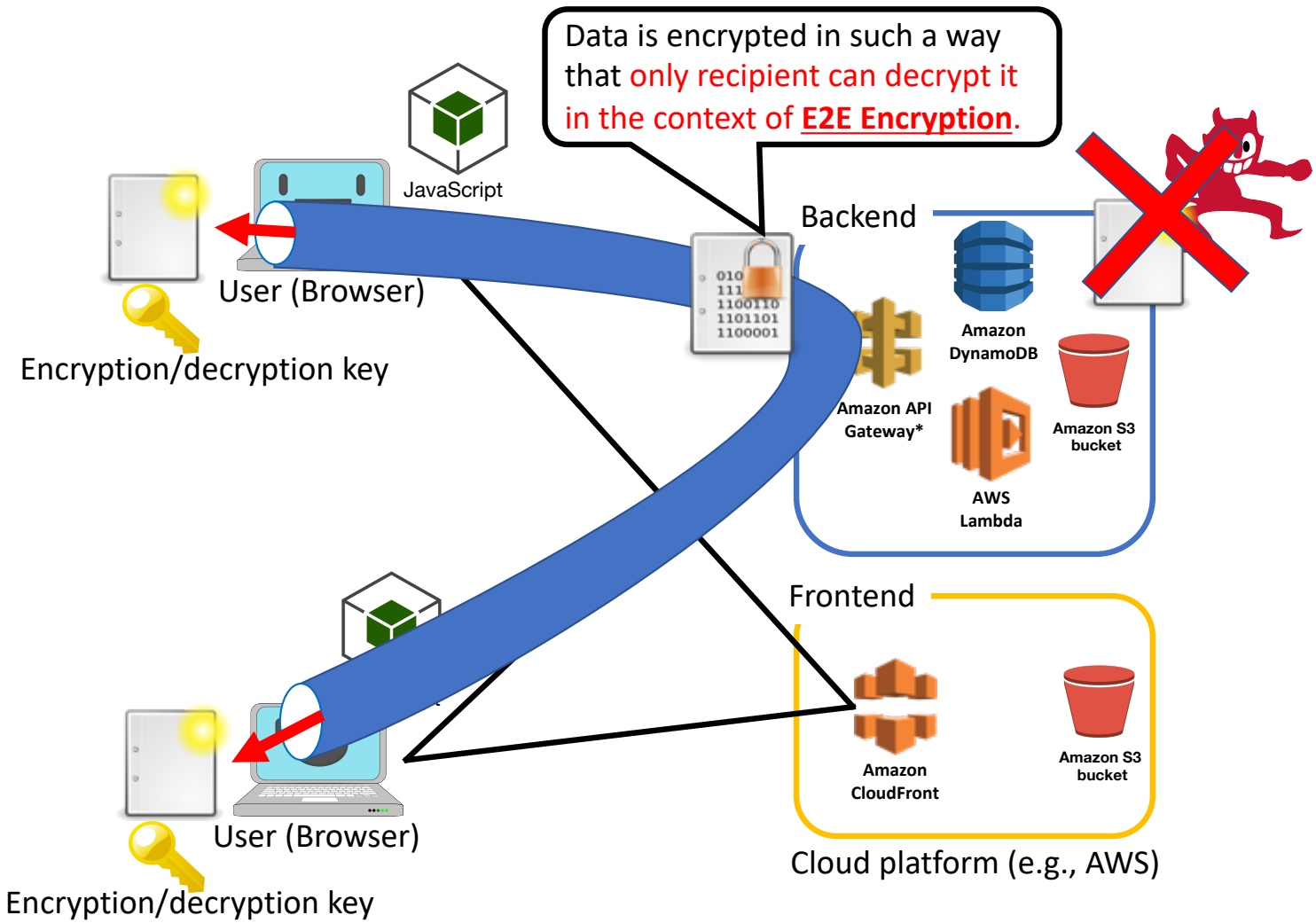


E2E セキュリティの観点からは、クラウドプラットフォーム自身ですら情報を取得可能であるべきではない



※ユーザがサーバとのみやり取りする場合、一方の主体 (End) はサーバとなる。

この例の場合、E2E セキュリティのためには、情報を共有する主体同士のみが復号可能なよう、やり取りするデータを全て暗号化する (E2E 暗号化)



E2E セキュリティを実現するには、構築するアプリケーション毎に「情報を共有する主体」を正しく定義する必要。

E2E セキュリティを実現しつつ、「事業者にとってリスクとなるような情報はなるべく不用意に取得しない」ことも重要。

Web App における End-to-End セキュリティ

改めて…

サービスで E2E セキュリティを考える意味

- ユーザプライバシの保護
- 事業者側のリスクの低減

では、Web アプリケーションにおけるエンドポイントとは？

- ユーザ側エンド:
  ⇒ JavaScript/HTML/CSS など「ブラウザ内で実行・レンダリングされる」Web フロントエンド要素[6]
- もう一方のエンド:
  ⇒ Web アプリケーション次第で変化

Web アプリケーションにおける E2E セキュリティ

少なくとも 1 つのエンドは Web フロントエンド要素
⇒ Web フロントエンドでセキュリティを担保する方法が必要

[6] 実際にユーザが触れる要素がエンドポイントであって、Web サービスにおいては、端末やブラウザはエンドポイントにならない

と、いうわけで、今回は「Web のための E2E セキュリティ」として JavaScript におけるデータの暗号化 (E2E 暗号化) の「さわり」を紹介します。

JavaScript で暗号を使ってみよう [基礎編]

–今回はお試しで AES–

AES (Advanced Encryption Standard) とは？

AES

共通鍵暗号・公開鍵暗号・ハッシュがどうのとか、そういう話は今回はしない。
とにかく「暗号化してみること」から始める。

JavaScript における暗号の利用環境

一般的な統合ライブラリは C で書かれた OpenSSL だが、JavaScript からは直接利用できない。

JavaScript から利用可能な暗号の統合ライブラリ:

- `WebCrypto API` (ブラウザ)[7]:
  W3C にて標準化が進む Web API (ブラウザのネイティブ API)。
- `Crypto` (Node.js)[8]:
  Node.js 環境にて利用可能な暗号ライブラリ (Node.js のネイティブ API)。OpenSSL のラッパー。
- `sjcl` (Node.js/ブラウザ)[9]:
  Stanford 大学暗号研究室で開発された pure JS なライブラリ。共通鍵暗号・ハッシュ等の実装が主。

[7] https://www.w3.org/TR/WebCryptoAPI/
[8] https://nodejs.org/api/crypto.html
[9] http://bitwiseshiftleft.github.io/sjcl/

今回は、高速動作が期待されるネイティブ実装な統合ライブラリ 2 つを例にとる。

- WebCrypto API
- Node Crypto

ブラウザでの暗号化: **WebCrypto API**

サーバでの暗号化: **Node.js Crypto**

ブラウザ・サーバ間での相互接続性の確認

しかしサーバで復号しているのであんまり意味がない。

補足: **API** が違うのがめんどくさい…

手前味噌だが、統合 **API** を使って楽をすると良い

ブラウザ同士での相互接続性の確認

API を通じて暗号化データをやり取りしてみる。
E2E Security!

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

引用文献

Appendix

This page is not counted.