

# サーバー運営上のセキュリティも大事

 : 「よし、認証周りもデータベース周りもばっちりだ!」

 : 「アプリを公開するぞ!!」

 : 「サーバーを建てる上でのセキュリティ、だいじょぶそ?」

 : 「あっつ...」

 : 「いくら素晴らしいソフトウェアを作ってもサーバーに侵入されちゃ終わりだからね」

# サーバーサイドセキュリティのきほんの「き」

## ファイアウォール

ファイアウォールとは、

企業などの内部ネットワークをインターネットを通して侵入してくる不正なアクセスから守るための「防火壁」

である。

パケットフィルタリングやアプリケーションゲートウェイなどの種類がある。

# ファイアウォールとは

## パケットフィルタリング

パケットのヘッダを解析して判断する

強力で柔軟だが、ものによっては設定が難しく、セキュリティホールが生まれる可能性がある

## アプリケーションゲートウェイ

サービスごとに認証する

細かい制御設定が難しい

## ファイアウォールとは

一般的なクラウドサービスでは、「パケットフィルタリング」の中でも「スタティックパケットフィルタ」と呼ばれる、IPアドレスやポート番号、プロトコルでアクセス制御を行う。

AWSではセキュリティグループがそれに当たる

## ファイアウォールのセットアップ例

- SSH (TCP, 22)は自宅のIPアドレス以外からの通信はブロック
- HTTP (TCP, 80)は全世界からアクセスOK

## サーバーへログインする際にはこれだけでは不十分

前のセクションで紹介した**ブルートフォースアタック**のような形でサーバーの管理者のユーザー名とパスワードを総当たりされてしまったら、サーバーに攻撃者が侵入してしまう可能性が...

→ 公開鍵認証でSSHアクセス

# 公開鍵認証とは

公開鍵認証とは、

公開鍵+秘密鍵の組み合わせを使って認証を行う方式

である。

鍵穴 (公開鍵)と鍵 (秘密鍵)のイメージ。この2つが合ってはじめて認証される。

## 公開鍵 (鍵穴)

- 誰にでも見られていいファイル
- .pubで終わるファイル
- サーバー側に渡してあげて自分専用の鍵穴を作るイメージ



## 秘密鍵 (鍵)

- 自分しか持っていてはいけないファイル
- パブリックキーとファイル名は同じだけ  
ど.pubとはつかない(混乱しやすいと思う  
で注意)

# 公開鍵認証とは

普段、みなさんがパソコンにログインするときはパスワードを使いますが、推測されてしまうなど、セキュリティ的にリスクがある。

一方で、公開鍵認証だと、秘密鍵を持っている人しか入れないので、秘密鍵が漏洩しない限り、セキュアな接続方法。（公開鍵認証は組み合わせが膨大すぎるので実質破ることはできない）

# SSHで公開鍵認証: 準備編

公開鍵・暗号鍵の組み合わせを生成

```
ssh-keygen -t rsa
```

`id_rsa` ファイル (秘密鍵) と `id_rsa.pub` ファイル (公開鍵) の2つが生成される

- **秘密鍵 ( `id_rsa` )**: クライアントに渡す
- **公開鍵 ( `id_rsa.pub` )**: サーバーに登録する

# SSHで公開鍵認証: サーバー編

`~/.ssh/authorized_keys` というファイルに公開鍵の中身を1つ1行づつで登録していく

# SSHで公開鍵認証: クライアント編

`~/.ssh/config` ファイルに下記の内容追記

```
Host ssh-demo
  HostName HOSTNAME
  User ubuntu
  IdentityFile ~/.ssh/id_rsa
```

`IdentityFile` の `~/.ssh/id_rsa` は秘密鍵の保存場所のパス

# SSHで公開鍵認証: クライアント編

秘密鍵をセキュアにする

## 最初の1文字

そのオブジェクトの種類

## 2-4文字目

ファイル所有者に対するアクセス権限

## 5-7文字目

ファイルの所有グループに対する権限

## 8-10文字目

他のユーザー等に対する権限

`ls -l` コマンドでファイルに対する権限を確認できる

オブジェクトの種類について

オブジェクト種類	意味
-	ファイル
d	ディレクトリ
l	シンボリックリンク

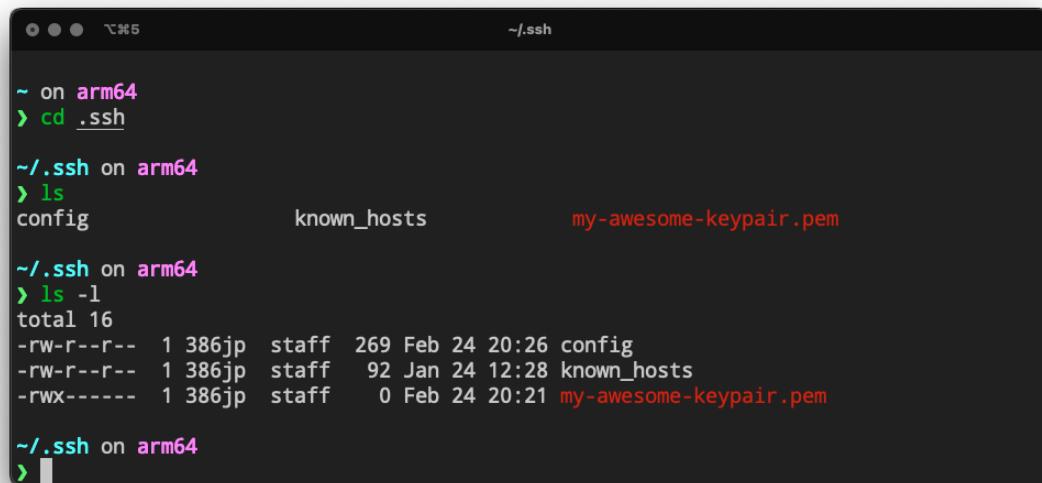
アクセス権限について

アルファベット	数字	意味
		権限の種類
r	4	読み取り
w	2	書き込み
x	1	実行

# SSHで公開鍵認証: クライアント編

秘密鍵をセキュアにする

`ls -l` コマンドでファイルに対する権限を確認できる



```
~ on arm64
> cd .ssh
~/ssh on arm64
> ls
config          known_hosts      my-awesome-keypair.pem

~/ssh on arm64
> ls -l
total 16
-rw-r--r--  1 386jp  staff  269 Feb 24 20:26 config
-rw-r--r--  1 386jp  staff   92 Jan 24 12:28 known_hosts
-rwx-----  1 386jp  staff     0 Feb 24 20:21 my-awesome-keypair.pem

~/ssh on arm64
> |
```

オブジェクトの種類について

オブジェクト種類	意味
-	ファイル
d	ディレクトリ
	シンボリックリンク

アクセス権限について

権限の種類		
アルファベット	数字	
r	4	読み取り
w	2	書き込み
x	1	実行

eg: `my-awesome-keypair.pem` ファイルはファイル所有者(自分)は読み取り、書き込み、実行ができるが、それ以外の人はなにもできない

# SSHで公開鍵認証: クライアント編

今回は、ファイル所有者(自分)しか読み取り、書き込み、実行ができないようにしたいので、下記のような権限にする

```
chmod 700 id_rsa
```

ちなみに、権限が適切でないとSSHしようとするときに怒られる

オブジェクトの種類について

オブジェクト種類	意味
-	ファイル
d	ディレクトリ
l	シンボリックリンク

アクセス権限について

権限の種類		
アルファベット	数字	
r	4	読み取り
w	2	書き込み
x	1	実行

# 公開鍵認証はGitHubのレポジトリにアクセスする際にも 使える！

特にプライベートレポジトリにアクセスする際、APIキーをGitHubで発行してそれをサーバーで使ってしまうと、意図しないレポジトリに対するアクセス権限も与えてしまう可能性が GitHubではデプロイキーという仕組みで、公開鍵認証の仕組みを使って特定のレポジトリのみアクセスできるようにできる

公開鍵認証の概念を使ってサーバー↔GitHubプライベートリポジトリの通信を確立  
GitHub側に鍵穴を作って、サーバー上の鍵を使ってリポジトリにアクセス

# GitHubで公開鍵認証: サーバー編

公開鍵・暗号鍵の組み合わせを生成

```
ssh-keygen -t rsa
```

Hint!

このとき、鍵のファイル名は `/home/ubuntu/.ssh/github_id_rsa` にしておく  
passphraseは入れないでおく

`id_rsa` ファイル (秘密鍵)と `id_rsa.pub` ファイル (公開鍵)の2つが生成される

- 秘密鍵 (`github_id_rsa`): サーバー側に持っておく
- 公開鍵 (`github_id_rsa.pub`): GitHubに登録

# GitHubで公開鍵認証: サーバー編

生成した鍵のうち、公開鍵をGitHubに登録するために表示しておく

```
cat ~/.ssh/github_id_rsa.pub
```

# GitHubで公開鍵認証: GitHub編

GitHubプライベートリポジトリ内、設定を開き、Deploy keysの設定を開く

The screenshot shows the GitHub repository settings page for the repository "386jp / flask-api-demo". The left sidebar has a vertical list of settings options: Options, Manage access, Security & analysis, Branches, Webhooks, Notifications, Integrations, Deploy keys (which is highlighted with a red border), Autolink references, Actions, and Secrets. The main content area is titled "Deploy keys" and contains the message "There are no deploy keys for this repository". Below this message is a link "Check out our guide on deploy keys to learn more." At the top right of the content area is a button labeled "Add deploy key". The top navigation bar includes links for Pull requests, Issues, Marketplace, Explore, and a search bar. The top right corner shows icons for Unwatch (with 1), Star (with 0), and Fork (with 0).

# GitHubで公開鍵認証: GitHub編

Add deploy keyから公開鍵を追加し、Add keyでキーを追加

Allow write accessのチェックボックスでサーバーからレポジトリに対する書き込みを制限することができる

[Deploy keys / Add new](#)

Title

Key

Begins with 'ssh-rsa', 'ssh-ed25519', 'ecdsa-sha2-nistp256', 'ecdsa-sha2-nistp384', or 'ecdsa-sha2-nistp521'

Allow write access

Can this key be used to push to this repository? Deploy keys always have pull access.

[Add key](#)

# GitHubで公開鍵認証: GitHub編

鍵が登録できました

## Deploy keys

Add deploy key



SSH

AWS

SHA256: LJ8weQJ961uYeJV1M4Qow1ztbPo2Yk92hIAglpHJv0Y

Added on Mar 24, 2021 by @386jp

Never used — Read-only

Delete

# GitHubで公開鍵認証: サーバー編

GitHubプライベートリポジトリにSSH経由でアクセスする。  
そのための接続設定を書き込んでおく

```
nano ~/.ssh/config
```

```
Host github github.com
  HostName github.com
  User git
  IdentityFile ~/.ssh/github_id_rsa
```

# GitHubで公開鍵認証: サーバー編

リポジトリをクローンしてくる

このとき、HTTPSではなく**SSH**でgit cloneしてくる

```
git clone git@github.com:386jp/ddos-web-2022.git
```

## 公開鍵認証にも限界はある

いくらファイアウォールの設定をっていても、いくら公開鍵認証を設定していても、少しの設定ミスやソフトウェアのバグで攻撃されてしまうかも…

もう少し段階を踏んでSSH接続をセキュアにする

→ 踏み台サーバーの登場

# 踏み台サーバーとは

basionサーバー / 踏み台サーバーとは、

目的のサーバーにログインするための中継サーバーのこと

である。

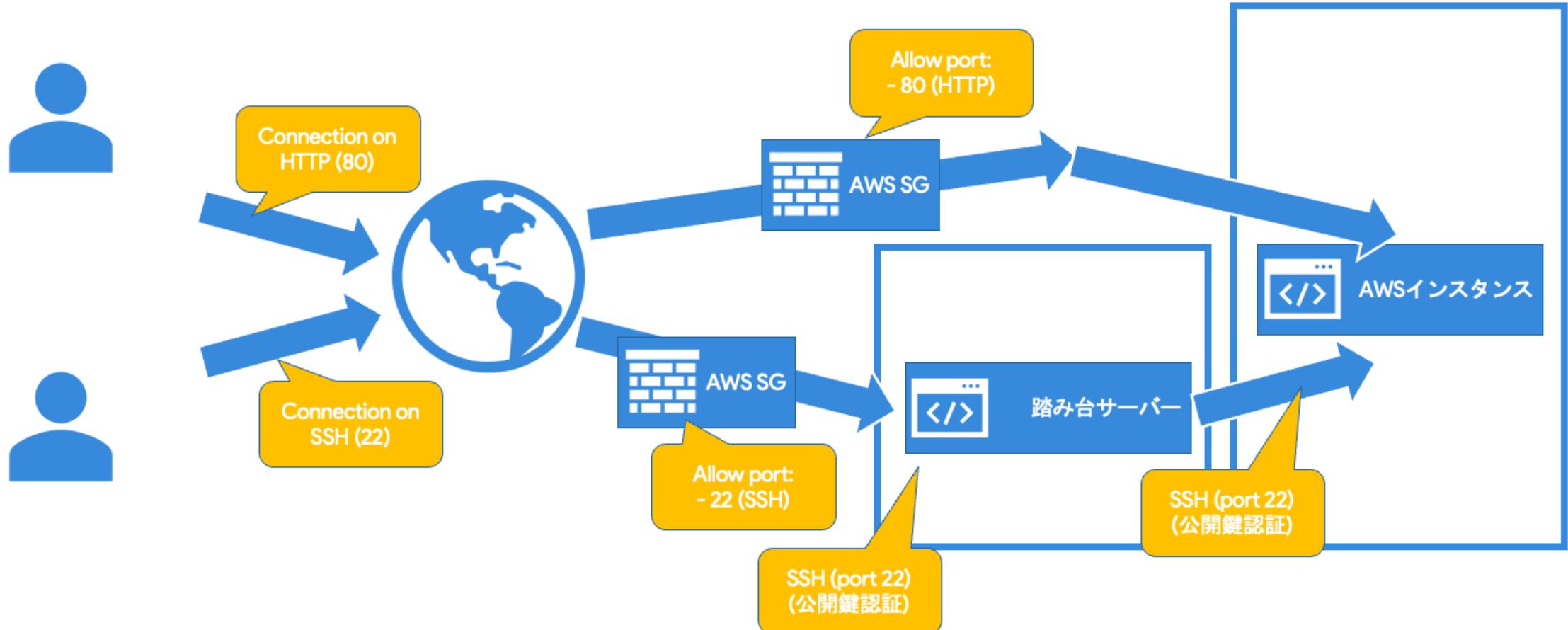
踏み台サーバーを使った環境の場合、

利用者はサーバーに対して直接アクセス (port 80, 443など)

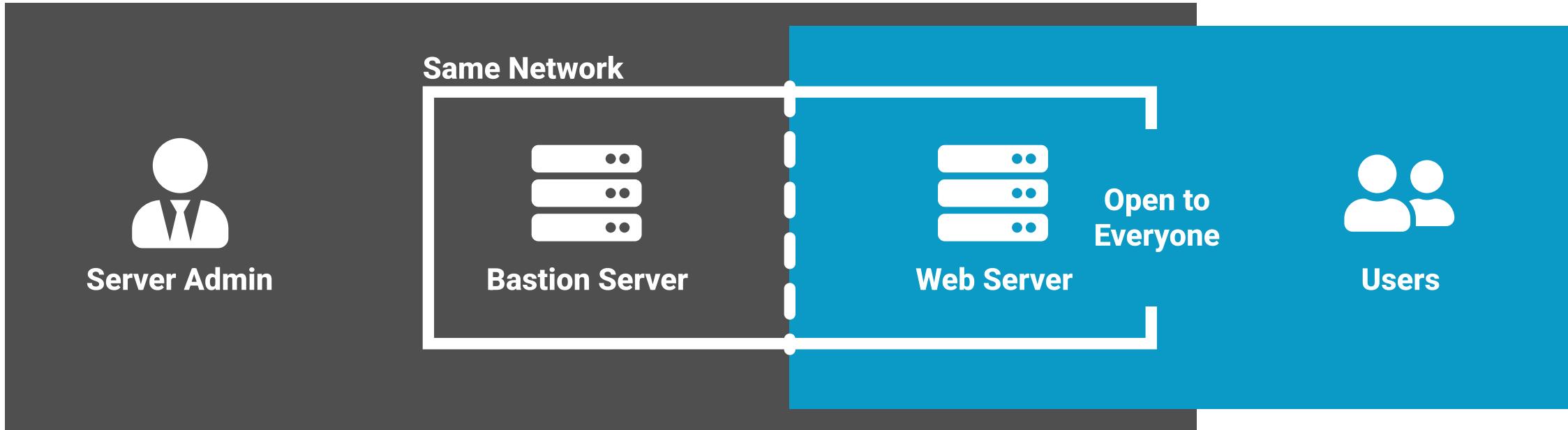
管理者は踏み台を経由してアクセス (port 22など)

となる

# 踏み台サーバーとは



# 踏み台サーバーの構築方法



- 踏み台サーバーとWebサーバーは同じプライベートネットワーク内に置く
- プライベートネットワーク内では踏み台からWebサーバーのSSH接続だけ許可
- 外部のネットワークからはWebサーバーの 80 , 443 ポートのみアクセス許可

# 踏み台サーバーの接続方法

```
### The Bastion Host
Host bastion-host-nickname
  HostName bastion-hostname

### The Remote Host
Host remote-host-nickname
  HostName remote-hostname
  ProxyJump bastion-host-nickname
```

ProxyJump を使って、踏み台サーバーを指定しておく

# なぜここまでSSHするために閑門を設ける必要があるのか

Webサーバーを公開していたら、そのサーバーのIPアドレスに対して攻撃が来る

特に、AWSなどのパブリッククラウドでサービスをホストしている場合、AWSのIPアドレスの範囲は決まっているので、攻撃者はそのIPアドレスの範囲を集中的に攻撃するので、余計に攻撃されやすい

## サーバー、ソフトウェア以外にも通信にも気を配るべし

パスワードなどの情報は特に、通信を暗号化する必要がある。

→ SSL通信

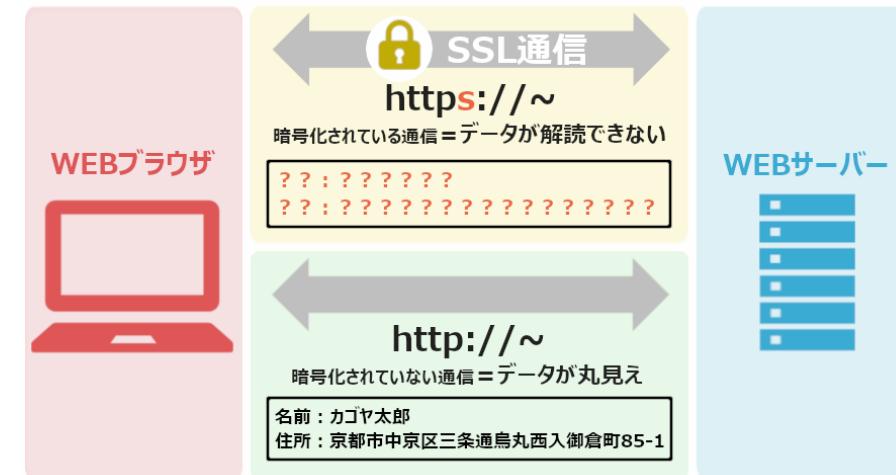
# SSL通信とは

SSL (Secure Sockets Layer) とは、

送受信しているデータを暗号化する通信手順である。

公開鍵認証の仕組みを使ってクライアントとサーバー間の通信を暗号化する

現在だと TLS (Transport Layer Security) という方式が使われているが、SSL という言葉が依然として馴染み深い。



## SSL通信のメリット

通信を暗号化したり、ドメインが本物だと証明できるので、

- データの盗聴
- データの改竄 (かいざん)
- なりすまし

これらを防ぐことができる

# SSL通信の概要

## 第1段階 (事前準備)

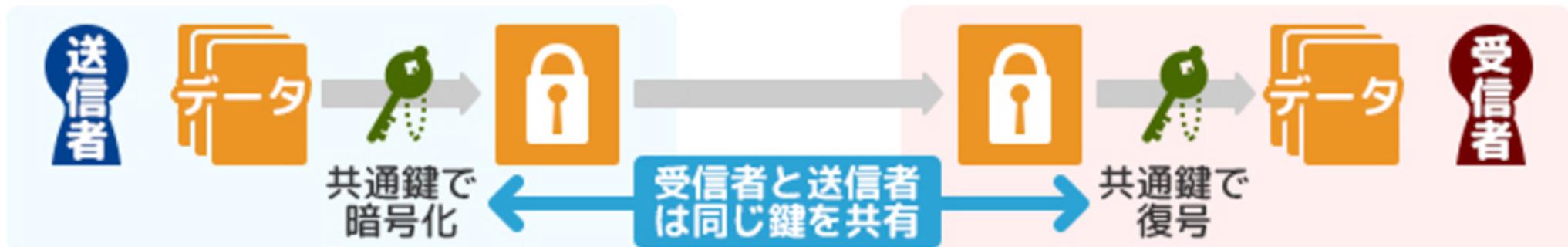


- 1) 受信者（サーバー）は送信者（ブラウザ）に「公開鍵」と「SSLサーバー証明書」を送付
- 2) 送信者（ブラウザ）は「公開鍵」を使って「共通鍵」を暗号化
- 3) 送信者（ブラウザ）は受信者（サーバー）に暗号化された「共通鍵」を送付
- 4) 受信者（サーバー）は「共通鍵」を「秘密鍵」で復号

※ここで言う復号とは、暗号を元に戻す仕組みのことです。

# SSL通信の概要

## 第2段階 (本番)



5) 受信者（サーバー）と送信者（ブラウザ）双方 上記でできた「共通鍵」を使って、本番データを暗号化と復号

## SSLサーバー証明書とは

SSLサーバー証明書によって、公開鍵がそもそも正しい鍵かどうかを証明することで、なりすましの防止ができます。

SSLサーバー証明書を発行する機関のことを認証局(CA= Certification Authority)と呼ぶ

# [Activity] SSL証明書を確認してみよう

386.jpの証明書を確認してみよう

まずは、URLの先頭が `https` で始まっていることを確認



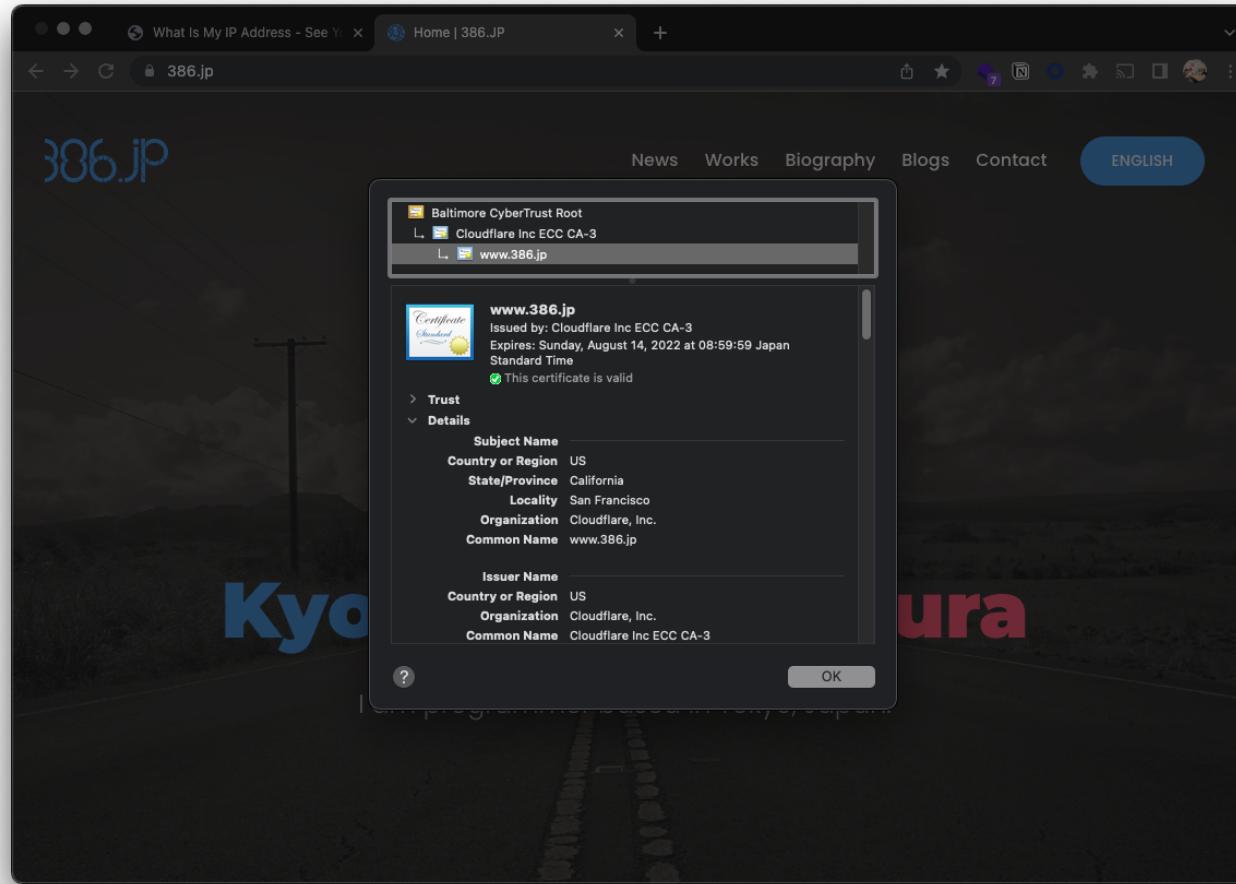
# [Activity] SSL証明書を確認してみよう



# [Activity] SSL証明書を確認してみよう



# [Activity] SSL証明書を確認してみよう



## 証明書以外の方法でもクライアントを保護する

近年では、SSL証明書とCA(認証局)の組合せ以外に、HSTS(HTTP Strict Transport Security)でこのドメインがSSL使っていることをブラウザ側に記憶させる技術や、Let's Encryptなどの無料SSL証明書配布サービス、常時SSLでないとGoogle等で検索順位が下がるなど、SSL技術に加え、サポートする形で様々な技術などが発達しており、情報の盗み見などの対策がされている

ここから発展的内容

# DDoSってなんだっけ

| 😕 脱データサイエンスオンラインセミナー

ではなく、、、

# DDoSとは

DDoS (Distributed Denial of Service)攻撃とは、

対象のウェブサイトやサーバーに対して大量のリクエストを送りつけてサービスを停止させる攻撃のこと

である。

# DoSとDDoS

**DoS (Denial of Service)**

攻撃元が1つ

**DDoS (Distributed Denial of Service)**

攻撃元が複数

一昔前まではサーバーが貧弱だったので、1つの攻撃元でも十分なダメージを与えることができたが、現在はサーバーも高性能化・耐障害性が上がったためかんたんにサービスを落とせなくなつたので、マルウェアなどで攻撃元のPCを増やした上で攻撃を行う方式になってきている

いわゆる田代砲は複数人でやってるので実質DDoS

## DDoSを防ぐには

- 攻撃元のIPアドレスからのアクセスを遮断する
- CloudFlareのようなDDoS対策サービスを囁ませる

ちなみに、CloudFlareの回しもんではありません

# CloudFlare繋がりで言うと…

CloudFlareはWAFというサービスも展開している

WAF (Web Application Firewall) とは、

Webアプリケーションの前面やネットワークに配置し、脆弱性を悪用した攻撃を検出・  
低減する対策のこと

である。

## WAF導入のメリット

WAFを導入することで、SQLインジェクションをはじめ、Webアプリに来る攻撃を防御できる

ただ、WAFを信用しすぎるのはよくないので、自分でもしっかり対策した上でお守り的な存在としてWAFも併用するのがおすすめ

# おさらい

- ORMを使ってSQLインジェクションからアプリを守る
- ユーザーを保護するために、認証はできればパスワードレス、もしくは二段階認証に
- パスワードのハッシュ化は必須!! 別途ブルートフォースアタックされにくい対策を
- プライマリーキーは面倒でも `uuid4` を使っていこう
- ファイアウォールや公開鍵認証、踏み台サーバーを使ってSSH接続をよりセキュアに
- SSL通信で通信を暗号化して通信を盗み見られたりなりすまされたりしないように
- DDoS対策やWAF設置をするとなおよし (CloudFlareおすすめ)