

SSL をはじめよう

「なんとなく」から「ちゃんとわかる！」へ

mochikoAsTech 著

2020-03-01 版 **mochikoAsTech 発行**

はじめに

2020 年 3 月 mochikoAsTech

本著を手に取ってくださったあなた、こんにちは！ あるいは、はじめて。『SSL をはじめよう』の筆者、mochikoAsTech です。

SSL は好きですか？ それとも怖いですか？ 本著を書くまで、筆者は「ちゃんと分かっているとは言えないので、SSL を迂闊にさわるのはなんだか怖い」と感じていました。

SSL は、エンジニアのごく身近なところにあります。たとえば「サイトをフル SSL 化する」「SSL 証明書を更新する」のような形で、誰しも一度は SSL と関わりをもったことがあるのではないでしょうか。しかし関わる機会が多い割に、「ちゃんと分かっているか」と言われるとちょっと自信がない、そんなエンジニアは、筆者を含め、きっと一定数いること思います。

Google が 2014 年ごろからさかんに提唱している「HTTPS everywhere」（直訳すると「どこでも HTTPS」）のもと、ウェブサイトの評価基準は「HTTP は普通、HTTPS にすれば安心」から、「HTTP は危険、HTTPS が普通」に変わってきました。ますます SSL と関わる機会が増えてきて、心のどこかで「SSL、ちゃんと分かりたいなあ…」と思っている。本著はそんな人のための一冊です。

理解度は「分かった！」「分かってなかった…」をくり返して、段々と上がっていくものです。まずは本著で、一緒に最初の「分かった！」まで進んでみましょう。

ちなみに本著「SSL をはじめよう」（以下 SSL 本）では、「DNS をはじめよう」（以下 DNS 本）で購入したドメイン名を使用します。DNS 本を読まずに SSL 本を読み進めていくと、第 2 章辺りで「ここで事前にあく抜きしておいた箇を取り出します」と言われて、「は？ あく抜きとかいつしてたの？！」という状態になります。「DNS は興味ないし面倒くさいんだけど…」という方も、できれば DNS 本を先にお読みいただいて、箇の下ごしらえ（＝ドメイン名の購入）を済ませた状態で SSL 本を開いてみてください。きっとその方が美味しくお召し上がりいただけます。なお SSL 本の第 1 章は、DNS 本を読んでいなくても問題ない内容ですので、とりあえずそのまま読み進めていただいても構いません。

またインフラやサーバに関する説明は、すでに「AWS をはじめよう」（以下 AWS 本）

で行なっていますので、本著では最低限にとどめています。本著で HTTPS のサイトを作ってみて、「インフラちょっと楽しいかも」と思われた方は、よかつたら後で AWS 本も召し上がってみてください。

DNS 本、AWS 本、そして SSL 本のはじめよう 3 部作は、「サーバやインフラは怖いものではなくすごく楽しいものなんだよ」ということを、かつての私のようなインフラ初心者へ伝えたくて書いたシリーズです。

読んで試して「面白かった！」と思ってもらえたなら、そしてインフラを前より少しでも好きになってもらえたなら何より嬉しいです。

想定する読者層

本著は、こんな人に向けて書かれています。

- よく分からぬまま SSL を使っている人
- 「サイトを HTTPS 化したいな」と思っている人
- 証明書の購入や設置の流れがいまいち分かっていない人
- SSL と TLS の関係性がよく分からぬ人
- SSL 証明書が一体何を証明しているのか知らない人
- これからシステムやプログラミングを学ぼうと思っている新人
- ウェブ系で開発や運用をしているアプリケーションエンジニア
- 「インフラがよく分からぬこと」にコンプレックスのある人

マッチしない読者層

本著は、こんな人が読むと恐らく「not for me だった…（私向けじゃなかった）」となります。

- SSL/TLS の通信を C 言語で実装したい人
- 「プロフェッショナル SSL/TLS」を読んで完全に理解できた人

本著の特徴

本著では実際にサーバを立てて SSL 証明書の設置を行い、HTTPS のサイトを作ってみます。手を動かして試してから仕組みを学べるので理解がしやすく、インフラ初心者でも安心して読み進められる内容です。Oracle Cloud の無料枠の中でサーバを立てて使用

しますので、サーバ代はかかりません。SSL 証明書代のみ、1,100 円（税込）かかります。
また SSL をめぐって実際にやってしまいがちな失敗、トラブルをとり上げて、

- こんな障害が起きたら原因はどう調べたらいいのか？
- 問題をどう解決したらいいのか？
- どうしたら事前に避けられるのか？

を解説するとともに、実際にコマンドを叩いて反復学習するためのドリルもついています。

本著のゴール

本著を読み終わると、あなたはこのような状態になっています。

- SSL 証明書がどんな役割を果たしているのか説明できる
- 証明書を買うときの手順が分かっている
- 意図せず「保護されていない通信」と表示されてしまったときの対処法が分かる
- 障害が起きたときに原因を調査できる
- 読む前より SSL が好きになっている
- SSL/TLS と併記されている「TLS」の意味が分かっている

免責事項

本著に記載されている内容は筆者の所属する組織の公式見解ではありません。

また本著はできるだけ正確を期すように努めましたが、筆者が内容を保証するものではありません。よって本著の記載内容に基づいて読者が行った行為、及び読者が被った損害について筆者は何ら責任を負うものではありません。

不正確あるいは誤認と思われる箇所がありましたら、必要に応じて適宜改訂を行いますので GitHub の Issue や Pull request で筆者までお知らせいただけますと幸いです。

<https://github.com/mochikoAsTech/startSSL>

目次

はじめに	3
想定する読者層	4
マッチしない読者層	4
本著の特徴	4
本著のゴール	5
免責事項	5
第1章 Oracle Cloud のアカウントを作ろう	11
1.1 サーバを立てる下準備	12
1.1.1 サイトを作るのにどうしてサーバがいるの？	12
1.1.2 サーバを立てるにはお金が必要？	13
1.1.3 なんで AWS じゃなくて Oracle のクラウドを使うの？	13
1.2 Oracle Cloud でアカウント登録	14
1.2.1 無料でアカウントを作ろう	14
1.2.2 〈トラブル〉 どうしても SMS が届かない！ そんなときは？	21
1.2.3 パスワードと支払情報の登録	21
1.2.4 Oracle Cloud のコンソールにサインイン	26
第2章 Oracle Cloud でサーバを立てよう	29
2.1 事前準備	30
2.1.1 Windows で RLogin をインストールする	30
2.1.2 Windows で SSH のキーペア（秘密鍵・公開鍵）を作成する	33
【コラム】 SSH の秘密鍵にパスフレーズは設定すべき？	37
2.1.3 Mac でターミナルを準備する	38
2.1.4 Mac で SSH のキーペア（秘密鍵・公開鍵）を作成する	40
【コラム】 ターミナルでコピー&ペーストするには？	41

2.2	コンピュートでサーバを立てる	42
2.2.1	OS は Oracle Linux 7.7 を使おう	43
2.2.2	作っておいた SSH の公開鍵を設置しよう	44
2.2.3	〈トラブル〉 "Out of host capacity." が起きたらどうすればいい？	44
2.2.4	Always Free ではなく無償クレジットの枠でサーバを立てよう	46
2.2.5	サーバが起動するまで待とう	48
	【コラム】 Oracle Cloud のコンピュートの金額計算方法	49
	【コラム】 Oracle Cloud と AWS はどっちが安い？	50
2.2.6	接続先となるサーバの IP アドレス	50
第 3 章	ウェブサーバの設定をしよう	53
3.1	サーバに SSH でログインしよう	54
3.1.1	Windows の RLogin を使ってサーバに入ってみよう	54
3.1.2	Mac のターミナルを使ってサーバに入ってみよう	64
3.2	ターミナルでサーバを操作・設定してみよう	66
3.2.1	プロンプトとは？	66
3.2.2	コマンドは失敗したときだけエラーを吐く	67
3.2.3	ターミナルを閉じたいとき	68
3.3	NGINX をインストールしよう	69
3.4	Firewalld で HTTP と HTTPS を許可しよう	71
3.5	SELinux を無効にしておこう	72
3.6	OS を再起動してみよう	75
3.6.1	ターミナルはなんのためにある？	76
3.7	なぜかサイトが見られない	77
3.7.1	サーバの手前にあるファイアウォールにも穴を空けよう	78
3.7.2	今度こそ HTTP でサイトを見てみよう	82
3.8	ドメイン名の設定をしよう	83
第 4 章	SSL 証明書を取得しよう	87
4.1	SSL 証明書にまつわる登場人物	88
4.2	秘密鍵 (startssl.key) を作ろう	88
	【コラム】 SSL 証明書の秘密鍵にパスフレーズは設定すべき？	89
4.3	CSR (startssl.csr) を作ろう	90
4.3.1	【ドリル】 CSR で入力すべきなのはクライアントの情報？	92
4.4	SSL 証明書の取得申請を出そう	92

4.5	DNS の設定をしよう	101
4.6	SSL 証明書をサーバに設置しよう	106
4.6.1	Windows で証明書と中間 CA 証明書をアップしよう	106
4.6.2	Mac で証明書と中間 CA 証明書をアップしよう	108
4.6.3	証明書を 1 ファイルにまとめよう	108
4.7	NGINX で HTTPS のバーチャルホストを作ろう	109
4.8	HTTPS でサイトを開いてみよう	111
	【コラム】ロードバランサーで SSL ターミネーションする方法もある	111
第 5 章	SSL/TLS について学ぼう	113
5.1	SSL/TLS とは?	114
5.1.1	SSL と TLS はどういう関係?	114
5.1.2	SSL イコール HTTPS ではない	114
5.2	「サイトを HTTPS 化する」とは何か?	115
5.3	どんなサイトでも必ず HTTPS にしなきゃだめ?	115
5.4	HTTP のままだと起きるデメリット	116
5.4.1	サイトが「安全でない」と表示されてしまう	116
5.4.2	Wi-Fi スポットでセッションハイジャックされる恐れがある	117
5.4.3	相対的に検索順位が下がる	117
5.4.4	周りが HTTPS になるとリファラが取れなくなる	118
5.5	HTTPS 化すると得られるメリット	118
5.5.1	表示速度が上がる	118
5.5.2	SameSite の変更に対応できる	119
5.5.3	重要情報のアタリが付けにくくなる	119
5.6	SSL 証明書とは	119
5.6.1	SSL サーバ証明書と SSL クライアント証明書	120
5.6.2	SSL 証明書はどんな場面で使われている?	120
5.7	SSL 証明書は異なる 3 つの仕事をしている	121
5.7.1	HTTPS の実際の流れ	122
5.7.2	ウェブページは 1 往復で表示されるわけじゃない	124
5.7.3	HTTP の混在コンテンツ（画像や CSS）があるとブロックされる	125
5.7.4	正当性を証明する中間 CA 証明書	127
5.7.5	ルート証明書はトラストストアにある	129
5.8	SSL 証明書はどうしてみんなに値段に差があるの?	131
5.9	同じ「SSL 証明書」という名前でも 3 つの種類がある	131

目次

5.9.1	3つの違いは何か？	131
5.9.2	DV 証明書	132
5.9.3	EV 証明書と OV 証明書	132
5.9.4	さよならグリーンバー	134
5.9.5	ブラウザベンダーによる EV 証明書の扱いの変化	134
5.10	その他の証明書	134
5.10.1	中間証明書	134
5.10.2	クロスルート証明書	134
5.11	どの証明書を買えばいい？	134
5.11.1	ワイルドカード証明書	134
5.11.2	www ありにリダイレクトしたいだけなのに www なしの証明書 もいるの？	134
5.11.3	コモンネームが*.example.com の証明書は example.com で使え る？	134
5.11.4	SANs	134
5.11.5	Let'sEncrypt	134
5.12	CDN と証明書	134
5.12.1	CDN を使ったら古い端末でサイトが見られなくなった	134
5.12.2	同じサーバで複数サイトを HTTPS 化したら古い端末で別サイ トが表示された	134
5.12.3	SNI Server Name Indication	134
あとがき		137
PDF 版のダウンロード		138
Special Thanks:		138
レビュー		138
参考文献・ウェブサイト		138
著者紹介		141

第1章

Oracle Cloud のアカウントを作 ろう

この章では Oracle Cloud というクラウドでアカウントを作ります。

SSL を理解するには、実際に手を動かしてやってみるのがいちばんです。実際に SSL 証明書を取得して、HTTPS のサイトを作ってみましょう。

HTTPS でサイトを作るのに必要な材料は次の 3 つです。

- ウェブサーバ
- ドメイン名
- SSL 証明書

まずは 1 つめのウェブサーバを立てるため、アカウント作成から始めましょう。

1.1 サーバを立てる下準備

HTTPSでサイトを作るのに必要な材料は次の3つです。

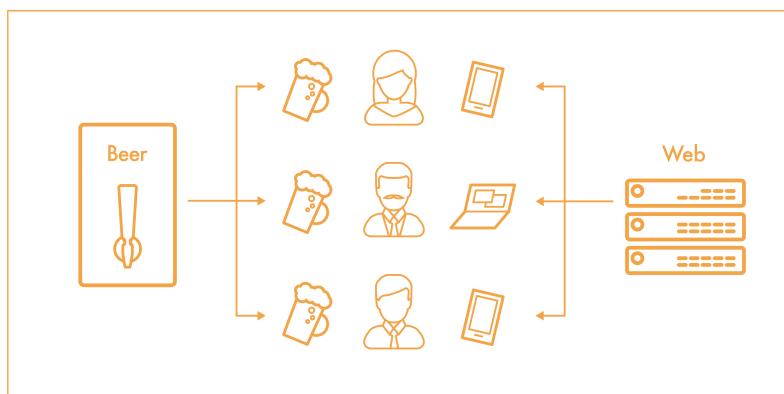
- ウェブサーバ
- ドメイン名
- SSL証明書

まずは1つめのウェブサーバを立てるための、下準備からはじめます。

1.1.1 サイトを作るのにどうしてサーバがいるの？

ところで、これからウェブサーバを立てようとしていますが…どうしてサイトを作りたいだけなのに、ウェブサーバが必要なのでしょう？

そもそもですが、サーバとはクライアントに対してサービスを提供するものです。居酒屋にあるビアサーバに「ビールをください」というリクエストを投げる…つまりコックを開の方へひねると、ビールというレスポンスが返ってきます。同様にあなたがブラウザでURLを入力したり、リンクをクリックしたりして、ウェブサーバに対して「ウェブページを見せてください」というリクエストを投げたら、ウェブページというレスポンスが返ってきます。（図1.1）



▲図1.1 ビアサーバもウェブサーバもリクエストしたらサービスが提供される

つまり、せっかくHTMLや画像でサイトのコンテンツを作っても、それを載せておくウェブサーバがなければ、サイトはあなたのパソコンの中でしか見られず、インターネット

トで公開できないのです。¹

というわけでウェブサイトを提供するために、まずはウェブサーバを立てましょう！

1.1.2 サーバを立てるにはお金が必要？

ウェブサイトを作るにはサーバが必要です。そしてサーバを立てるには、普通はお金がかかります。ですがオラクルがやっている「Oracle Cloud（オラクル クラウド）」というサービスなら、なんと有効期限なしでずっと無料で使える「Always Free」という枠があります。「Always Free」の範囲内であれば、サーバも無料で立てて使えるので今回はそれを使いましょう。

オラクルがやっているクラウド、と言われても、そもそもクラウドがなんだか分からないといまいちピンと来ないかもしれません。あなたが「ウェブサイト作りたいなあ…だからサーバが必要だ！」と思ったとき、**自分でサーバを買って自分で管理しなければいけないのがオンプレミスで、従量課金ですぐに使って性能や台数の増減も簡単にできるのがクラウドです。**

Oracle Cloud とはオラクルがやっているクラウドなので、ブラウザでぽちぽちとスペックを選んでいくだけで、すぐにサーバが使えます。

1.1.3 なんで AWS じゃなくて Oracle のクラウドを使うの？

クラウドは Oracle Cloud だけではありません。かの有名な AWS こと Amazon Web Services や、Google の Google Cloud Platform²、Microsoft の Azure（アジュール）³、その他にも国内クラウドとしてさくらインターネットがやっているさくらのクラウド⁴、お名前.com でお馴染み GMO グループの GMO クラウド⁵などたくさんあります。

2019年11月時点、クラウド市場では AWS がシェア約40%でトップを独走中⁶です。そのため仕事で AWS を使ったことがある、あるいはこれから使う予定だ、というエンジニアも多いと思います。

*1 サーバについては、はじめようシリーズの2冊目、「AWSをはじめよう」の「CHAPTER1 インフラとサーバってなに？」で、より詳しく解説しています。仮想サーバと物理サーバ、クラウドとオンプレミス、ホストサーバとゲストサーバなどサーバ周りの用語をもう少し理解したい！という方はそちらも併せて読んでみるのがお勧めです

*2 <https://cloud.google.com/>

*3 <https://azure.microsoft.com/ja-jp/>

*4 <https://cloud.sakura.ad.jp/>

*5 <https://www.gmocloud.com/>

*6 IaaS + PaaS クラウド市場、AWSの首位ゆるがず。AWS、Azure、Google、Alibaba の上位4社で市場の7割超。2019年第3四半期、Synergy Research Group — Publickey https://www.publickey1.jp/blog/19/iaaspaaawsazuregooglealibaba4720193synergy_research_group.html

しかし最近は、Alibaba Cloud や Tencent Cloud といった中国のクラウド事業者も追い上げを見せています。こうした新興のクラウドは、先に行く AWS を見て学んだ上で生まれてきているだけあって、よりスマートな作りになっているのがいいところです。

たくさんのクラウドがある中でどこを選ぶのか、その理由は、本来であれば使う人やその上で動かすサービスによって異なるはずです。あなたが動かしたいサービスには、一体どのクラウドが適しているのでしょうか？

本著では次の 2 つを目的としていますので、それに適した Oracle Cloud で学びを進めていきたいと思います。

- SSL 証明書を自分で取得して設置する一通りの流れを試したい
- お金をかけずに無料で試したい

1.2 Oracle Cloud でアカウント登録

まずは Oracle Cloud のアカウントを作りますので次の 2 つを用意してください。

- クレジットカード
- SMS 受信が可能な携帯電話（電話番号認証で使用するため）*7

なお Oracle Cloud を利用する際は、前述のとおり Always Free という無料枠*8があります。

1.2.1 無料でアカウントを作ろう

「Oracle Cloud 無料」で検索（図 1.2）したら、いちばん上の [Oracle Cloud Free Tier | Oracle 日本]*9をクリックします。

*7 ショートメッセージサービスの略。宛先に電話番号を指定してメッセージを送れるサービス

*8 期限なしでずっと無料ですが、無料で利用できる範囲は決まっていて、何をどれだけ使っても無料という訳ではありませんので注意してください。Always Free の他に、30 日間だけ有効な 300 ドル分の無償クレジットも付いてきますので、Always Free の範囲外のサービスはそちらで試せます。詳細は <https://www.oracle.com/jp/cloud/free/> を確認してください

*9 <https://www.oracle.com/jp/cloud/free/>

1.2 Oracle Cloud でアカウント登録



▲図 1.2 「Oracle Cloud 無料」で検索

Oracle Cloud Free Tier のページが表示されたら、[今すぐ始める（無償）] をクリックします。（図 1.3）

第1章 Oracle Cloud のアカウントを作ろう



▲図 1.3 [今すぐ始める (無償)] をクリック

「Oracle Cloud へのサインアップ」と表示されたら、[電子メール・アドレス] と [国/地域] を入力して、使用条件を確認した上で [次] をクリックしましょう。(図 1.4) 後で分からなくなないように、登録した項目を表 1.1 にメモしておきましょう。

▼表 1.1 Oracle Cloud に登録した情報

項目	例	あなたが登録した情報
電子メール・アドレス	startdns.01@gmail.com	
国/地域	日本	



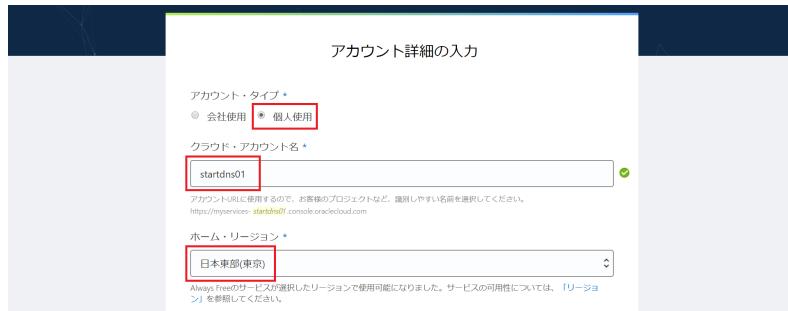
▲図 1.4 入力したら [次] をクリック

次は「アカウント詳細の入力」です。(表 1.2) 今回は仕事ではなく個人での利用ですので〔アカウント・タイプ〕は〔個人使用〕を選択してください。〔クラウド・アカウント名〕には任意のアカウント名を入力します。〔クラウド・アカウント名〕には英字小文字と数字のみ使えます。記号や英字大文字は使えないで注意してください。筆者は startdns01 にしました。この〔クラウド・アカウント名〕は、後で管理画面にサインインするときのアカウント URL になります。(図 1.5)

〔ホーム・リージョン〕は〔日本東部(東京)〕を選択してください。Oracle Cloud は世界の各地域にデータセンターを所有しており、サーバはそのデータセンターの中で元気に動いています。この〔ホーム・リージョン〕とは、**各地域の中でどこを使うか？を指定するものです**。ウェブサイトにアクセスするとき、パソコンのある場所からサーバまで物理的に距離が遠いと、それだけ通信にも時間がかかるって応答時間も遅くなります。日本国内向けにウェブサイトを開設する場合は、基本的にこの〔日本東部(東京)〕のリージョンを選びましょう。ただし Oracle Cloud のサービスによってはまだ東京リージョンが使えないものもあります。その場合は次点として「米国東部(アッシュバーン)」を選択してください。

▼表 1.2 Oracle Cloud に登録した情報

項目	例	あなたが登録した情報
アカウント・タイプ	個人使用	
クラウド・アカウント名	startdns01	
ホーム・リージョン	日本東部(東京)	



▲図 1.5 [クラウド・アカウント名] には好きな名前を入力

続いて名前や住所を入力していきます。入力内容は日本語表記で構いません。個人利用なのですが「部門名」が必須であるため、ここでは「個人」と入力しておきましょう。「名」・「姓」・「部門名」・「住所」・「市区町村」・「都道府県」・「郵便番号」をすべて入力できましたか？（図 1.6）

The screenshot shows a registration form with several input fields highlighted by red boxes:

- 名 *: 猫村
- 姓 *: もち子
- 部門名 *: 借入
- 役職: (empty)
- 住所 *: 新宿四丁目1番6号
JR新宿ミライナタワー
- 市区町村 *: 新宿区
- 都道府県 *: TOKYO
- 郵便番号 *: 160-0022
- 国/地域: 日本

▲図 1.6 名前や住所を入力

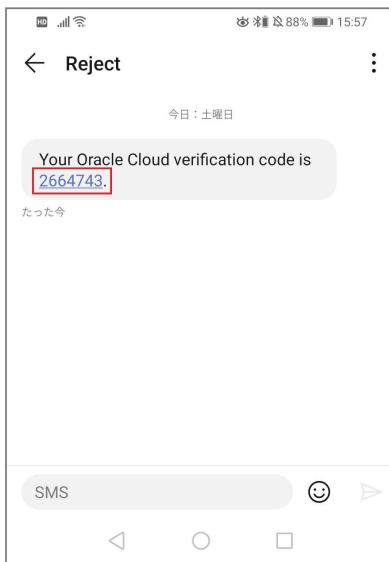
では最後に「モバイル番号」です。国番号は「日本(81)」を選択して、自分の携帯電話番号を入力します。このとき電話番号の先頭の 0 は不要です。例えば「090-〇〇〇〇-〇〇〇〇」という携帯電話番号であれば「90-〇〇〇〇-〇〇〇〇」と入力してください。携帯電話番号を入力したら【次: モバイル番号の確認】をクリックしてください。(図 1.7)

The screenshot shows a form for entering a mobile phone number:

- モバイル番号 *: 日本(81) [dropdown menu]
90 [input field]
- 下記のモバイル番号および電子メール startdn10@gmail.com に基づく検証コードがすでにある場合、[ここをクリックしてコードを確認してください。](#)
- 次: モバイル番号の確認 [button]
- サポートが必要ですか。ご連絡ください: [チャット・サポート](#)

▲図 1.7 携帯電話の番号を入力

数分以内に [Your Oracle Cloud verification code is 〇〇〇〇〇〇〇〇.] と書かれた SMS が届きます。(図 1.8)



▲図 1.8 コードの書かれた SMS が届いた

SMS で届いた「〇〇〇〇〇〇〇〇」の数字を [コード] に入力して、[コードの確認] をクリックします。(図 1.9)



▲図 1.9 SMS で届いた数字を [コード] に入力して [コードの確認] をクリック

1.2.2 〈トラブル〉 どうしても SMS が届かない！ そんなときは？

電話番号を入力したのに SMS が届かないときは、まず自分が契約している携帯キャリアの迷惑メール設定で、SMS をスパムとしてはじく設定をしていないか確認してみましょう。たとえば海外の事業者から送信された SMS を拒否する設定になっていたり、海外からの着信を拒否する設定になっていると、SMS が届かないことがあるようです。^{*10}

ちなみに筆者の場合は、特に設定変更をせず同じ番号で 2 回試してみたのですが、最初は届かず、もう 1 回試してようやく届きました。

迷惑メールの設定を確認して何回か試して、それでも SMS が届かなかったら、ページ下部の [サポートが必要ですか。ご連絡ください: チャット・サポート] からサポートにチャットで問い合わせてみましょう。残念ながら英語でしか対応してもらえませんが、"I am trying to register on Oracle Cloud. But I can't receive SMS. What should I do?" (アカウント登録しようとしてるけど SMS が届かないの、どうしたらいい？) という感じで聞いてみると、「じゃあ登録情報をこのチャットで教えて。そうしたらこちらでコードを発行して、このチャットで伝えてあげる」(意訳) という感じでサポートしてもらえます。

1.2.3 パスワードと支払情報の登録

正しいコードが入力できたら、[パスワードの入力] が表示されます。[パスワード] と [パスワードの確認] を入力して、[次: 支払情報] をクリックします。(図 1.10)

^{*10} 「Oracle Cloud の SMS は海外の事業者から届く」という確証がある訳ではないです。あくまで SMS が届かないときによくある話と思ってください



▲図 1.10 パスワードを入力して [次: 支払情報] をクリック

パスワードを入力すると、今度は「支払情報」のページが表示されます。(図 1.11) 繰り返しあ伝えしているとおり、Oracle Cloud には Always Free という無料枠があります。本著では基本的にこの無料枠の範囲内で Oracle Cloud を使っていくつもりですが、それでもクレジットカードは登録しておく必要があります。

なおページに記載されているとおり、この後、管理画面で「アカウントのアップグレード」という作業をしない限り、請求は発生しませんので安心してカード情報を登録してください。[クレジット・カード詳細の追加] をクリックします。



▲図 1.11 [クレジット・カード詳細の追加] をクリック

[ご注文者様情報] はそのまま変更不要です。[カード情報] の [カードの種類] を選択し、[カードの番号]・[有効期限]・[CVN] を入力したら [Finish] をクリックします。
(図 1.12)*¹¹

*¹¹ Oracle Cloud では、クレジットカード登録時に「1 ドル認証」と呼ばれる認証方法で、そのクレジットカードが決済可能かをチェックしています。クレジットカードによってはこの 1 ドル認証を不審な決済と判断して通さないため、それによってエラーが発生することがあります。その場合は別のクレジットカードで試すか、Oracle Cloud のチャット・サポートで問い合わせてみてください

カード情報 ▲

カードの種類 *

Visa Mastercard
 Amex JCB

カードの番号 *

有効期限 *

CVN *

このコードは、クレジットカードの裏面または表面に印字されている3桁または4桁の番号です。

Finish

▲図 1.12 カード情報を入力して [Finish]

[クレジット・カード詳細をご提供いただきありがとうございます。] と表示（図 1.13）されたら、支払い情報の登録は完了です。Oracle Cloud の Service Agreement^{*12}を確認した上で、チェックボックスにチェックを入れて、[サインアップの完了] をクリックします。

^{*12} <https://www.oracle.com/goto/oraclecsa-jp-en>

1.2 Oracle Cloud でアカウント登録



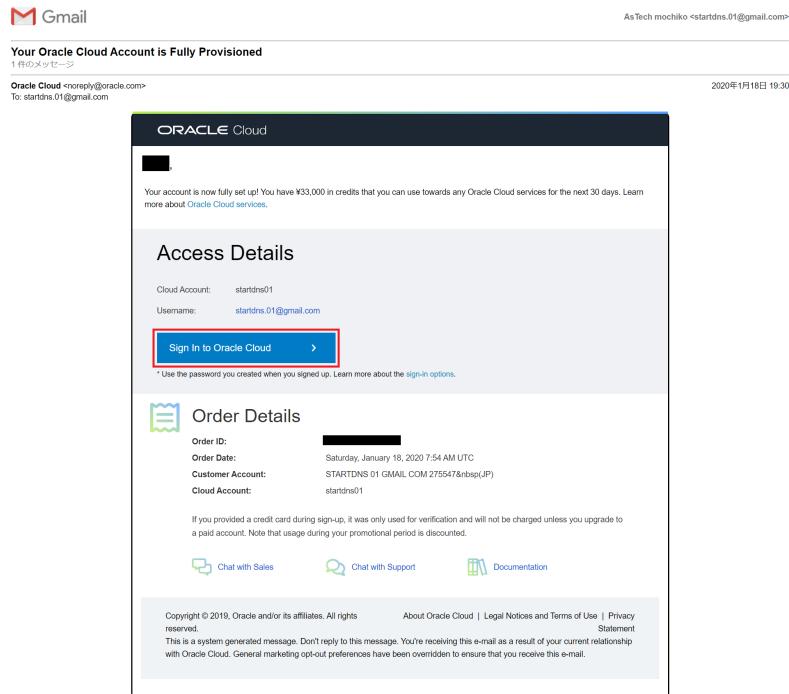
▲図 1.13 チェックを入れて [サインアップの完了] をクリック

これでアカウント登録の手続きはおしまいです。[アカウントの設定が完了するまでお待ちください。] と表示（図 1.14）されます。準備が整うとサインイン画面にリダイレクトされますが、この [アカウントの設定が完了するまでお待ちください。] の画面でかなり時間がかかるので、一度ブラウザを閉じてしまって構いません。頑張った自分を褒めて、一旦休憩にしましょう。



▲図 1.14 アカウント登録の手続きはおしまい

数時間後^{*13}、[Your Oracle Cloud Account is Fully Provisioned] という件名で、準備完了を知らせるメールが届きます。メールの [Sign In to Oracle Cloud] をクリックしましょう。(図 1.15)



▲図 1.15 数時間後、準備完了を知らせるメールが届く

1.2.4 Oracle Cloud のコンソールにサインイン

メールの [Sign In to Oracle Cloud] をクリックすると、コンソールへのサインイン^{*14}画面が表示されます。(図 1.16) [ユーザー名] には先ほど登録したメールアドレスを入力します。^{*15} [パスワード] を入力して、[サイン・イン] をクリックしてください。

*13 筆者の場合は、メールが届くまで 2 時間半かかりました

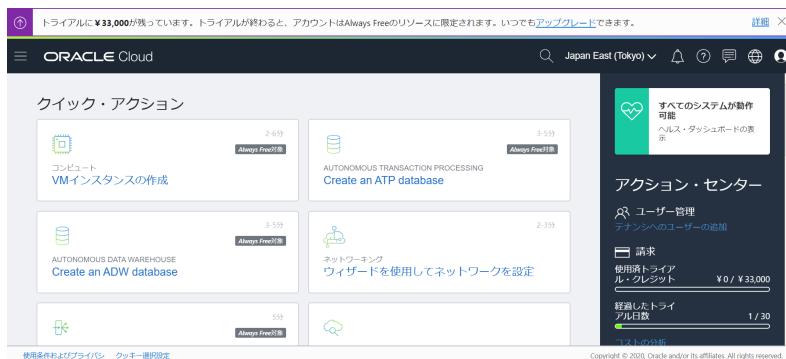
*14 日本語だとログインの方が馴染みがあるかも知れませんが、サインインはログインと同じ意味です

*15 メールにも書いてありますが、ここでの [ユーザー名] とは [クラウド・アカウント名] (筆者の場合は startdns01) ではなく、[メールアドレス] のことです。紛らわしいのでご注意ください



▲図 1.16 [ユーザー名] と [パスワード] を入力して [サイン・イン]

おめでとうございます！ コンソールにサインインできました。



▲図 1.17 コンソールにサインインできた！

なお今後、コンソールにサインインしたくなったら、いちいち Oracle Cloud からのメールを探してリンクを踏む必要はありません。まずは Oracle のトップページ^{*16}を開いて、右上の人マークから [クラウドにサインイン] をクリックしましょう。

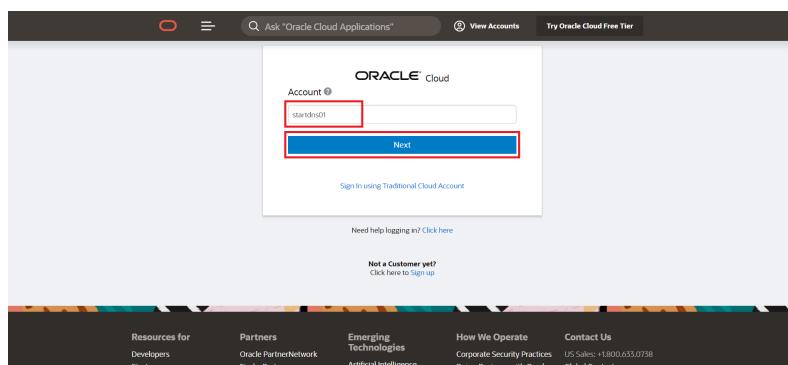
*16 <https://www.oracle.com/jp/>

第1章 Oracle Cloud のアカウントを作ろう



▲図 1.18 右上の人マークから [クラウドにサインイン] をクリック

サインインのページ^{*17}で [Account] の欄にクラウド・アカウント名^{*18}を入力して [Next] をクリックすれば、メールのリンクを踏んだときと同じ [サイン・イン] のページにたどり着けます。あとは同じように [ユーザー名] にはメールアドレスを、[パスワード] にはパスワードを入力して、[サイン・イン] をクリックするだけです。



▲図 1.19 [Account] の欄にクラウド・アカウント名を入力して [Next] をクリック

*17 <https://www.oracle.com/cloud/sign-in.html>

*18 筆者の場合は startdns01 です。アカウント登録時に、あなたの [クラウド・アカウント名] をメモしているはずですでの、数ページ戻って確認してみましょう

第2章

Oracle Cloud でサーバを立てよう

この章では実際に Oracle Cloud でサーバを立てます。
インフラエンジニアのお仕事体験みたいできっと楽しいですよ！

2.1 事前準備

2.1.1 Windows で RLogin をインストールする

Windows のパソコンを使っている方は、サーバを立てる前に「ターミナル」と呼ばれる黒い画面のソフトをインストールしておきましょう。サーバに接続するときにはこのターミナルを使うのですが、ターミナルのソフトには色々な種類があります。

- RLogin (<http://nanno.dip.jp/softlib/man/rlogin/>)
- Poderosa (<https://ja.poderosa-terminal.com/>)
- Tera Term (<https://ja.osdn.net/projects/ttssh2/>)
- PuTTYjp (<http://hp.vector.co.jp/authors/VA024651/PuTTYkj.html>)



▲図 2.1 RLogin

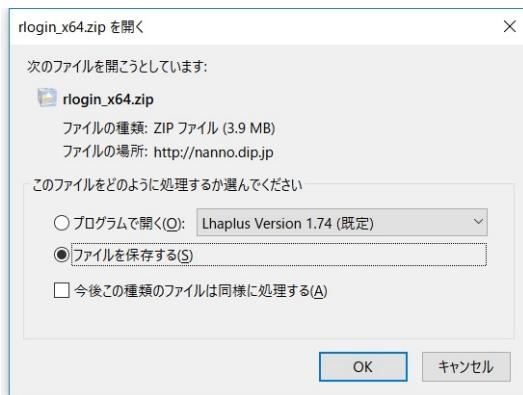
本著ではいちばん上の RLogin (図 2.1) を使って説明していくので、特にこだわりがなければ RLogin を使うことをお勧めします。RLogin の「実行プログラム (64bit)^{*1}」(図 2.2) の URL、http://nanno.dip.jp/softlib/program/rlogin_x64.zip をクリックしてください。

^{*1} もしパソコンの Windows が 32bit 版だった場合は「実行プログラム (32bit)」の URL をクリックしてください。



▲図 2.2 「実行プログラム (64bit)」の URL をクリックしてダウンロード

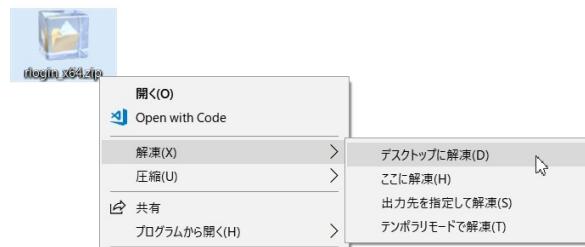
ダウンロードした ZIP ファイルを保存（図 2.3）します。保存場所はどこでも構いませんが、後でどこに置いたか分からなくなりそうな人はデスクトップに保存しておきましょう。



▲図 2.3 「ファイルを保存する」でパソコンに保存

デスクトップの ZIP ファイル (rlogin_x64.zip) を右クリック（図 2.4）して、[解凍>デスクトップに解凍]*2をクリックします。

*2 ZIP ファイルを右クリックしても「解凍」が見当たらないときは、圧縮・解凍の定番ソフトである Lhaplus をインストールしましょう。 <https://forest.watch.impress.co.jp/library/software/lhaplus/>



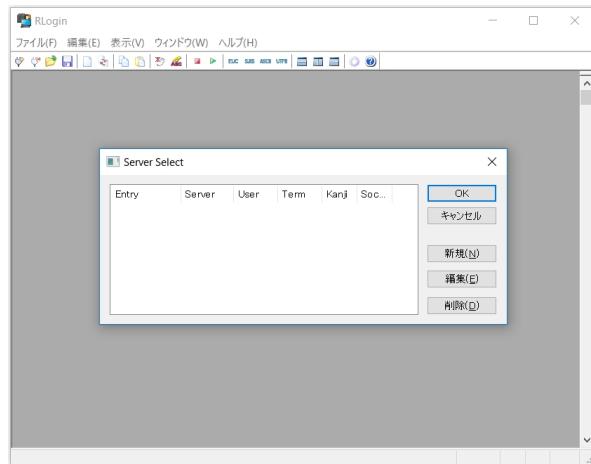
▲図 2.4 ZIP ファイルを右クリックして解凍>デスクトップに解凍

解凍したら、デスクトップにできた「rlogin_x64」というフォルダの中にある「RLogin.exe」*3（図 2.5）をダブルクリックすれば RLogin が起動（図 2.6）します。



▲図 2.5 RLogin.exe をダブルクリック

*3 フォルダの中に RLogin はあるけど RLogin.exe なんて見当たらない…という場合、ファイルの拡張子が非表示になっています。この後も拡張子を含めてファイル名を確認する場面が何度かでできますので、表示されていない人は「拡張子 表示」で Google 検索して、拡張子が表示されるように設定変更しておきましょう。

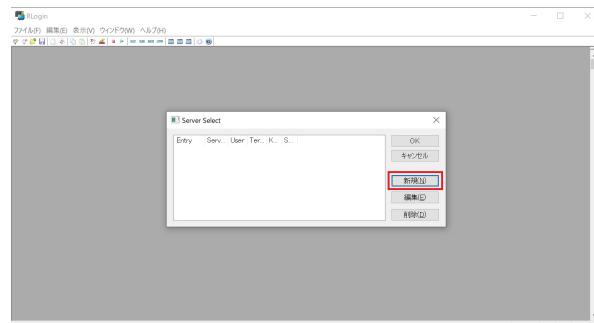


▲図 2.6 RLogin が起動した

これで RLogin のインストールは完了です。

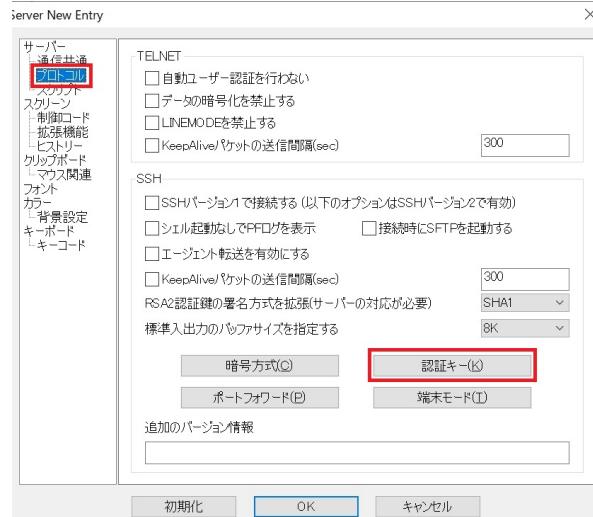
2.1.2 Windows で SSH のキーペア（秘密鍵・公開鍵）を作成する

Windows の方は、続いて起動した RLogin で [新規 (N)] をクリックします。



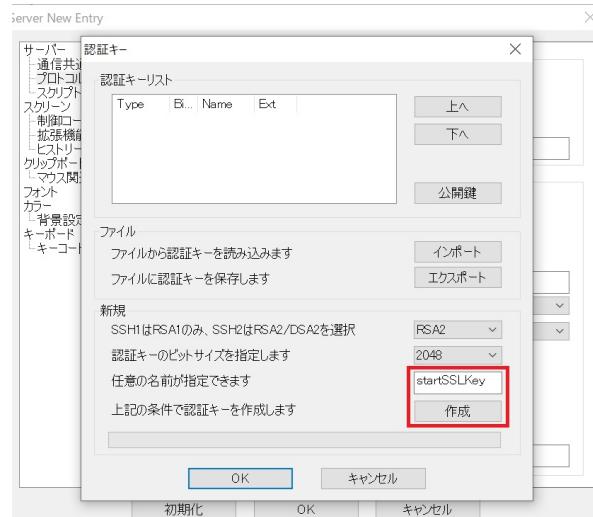
▲図 2.7 [新規 (N)] をクリック

左メニューの [プロトコル] を選択して、[認証キー (K)] をクリックします。



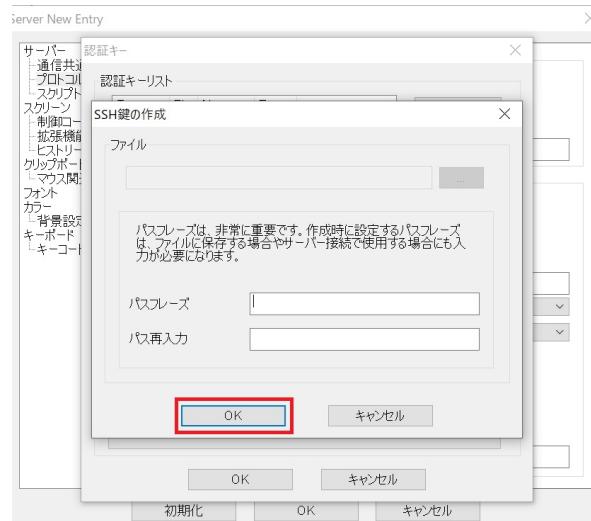
▲図 2.8 [プロトコル] を選択して [認証キー (K)] をクリック

[任意の名前が指定できます] に [startSSLKey] を入力して、[作成] をクリックします。



▲図 2.9 [startSSLKey] を入力して [作成] をクリック

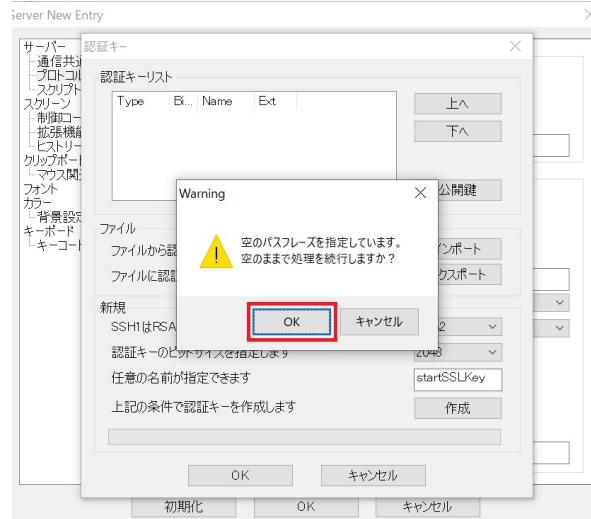
[パスフレーズ] と [パス再入力] には何も入力せず、[OK] をクリックします。^{*4}



▲図 2.10 何も入力せず [OK] をクリック

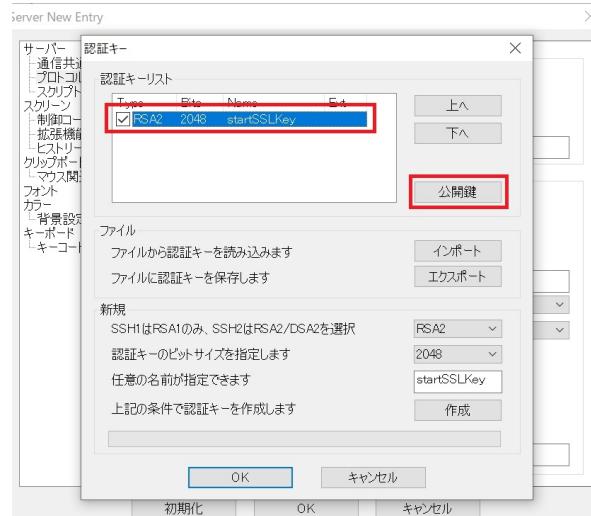
[空のパスフレーズを指定しています。空のままで処理を続行しますか？] と表示されますが、そのまま [OK] をクリックします。

^{*4} 「p@\$sw0rd」 や「@dm1ni\$trat0r」 のように、ひとつの単語でできているのがパスワードです。それに対して「This 1s P@s\$ Phrase.」のように空白を挟んだ文章（フレーズ）で構成されているのがパスフレーズです



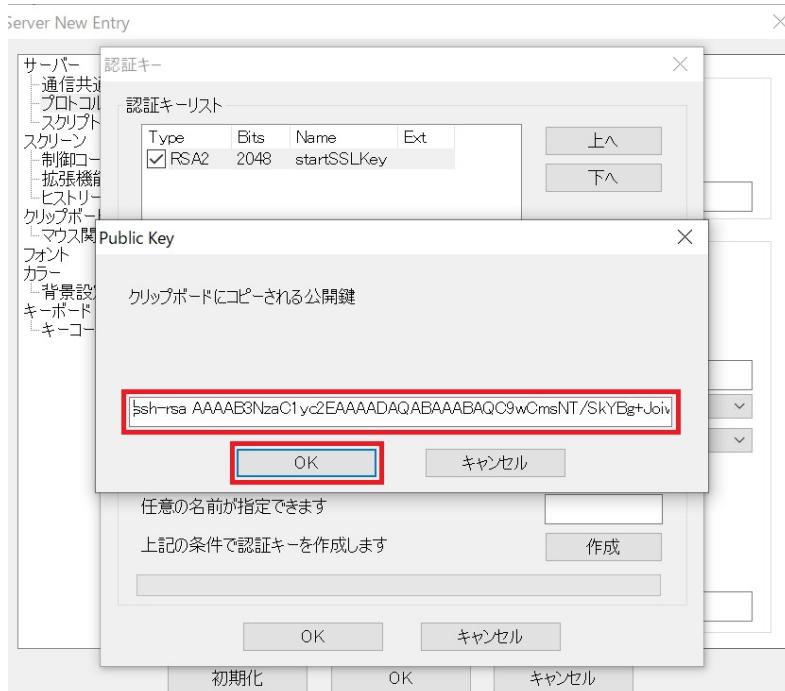
▲図 2.11 [OK] をクリック

【認証キーリスト】に、いま作った【startSSLKey】が表示されたら、キーペア（秘密鍵・公開鍵）が無事できています。【公開鍵】をクリックしてください。（図 2.5）



▲図 2.12 【startSSLKey】が表示されたら【公開鍵】をクリック

この後すぐに使いますので、[クリップボードにコピーされる公開鍵] で表示された公開鍵（ssh-rsa から始まる文字列）をまるごとコピーして、メモ帳などにペーストしておきましょう。公開鍵をメモしたら [OK] をクリックして閉じます。



▲図 2.13 表示された公開鍵（文字列）はまるごとコピーしてメモ帳にペーストしておこう

あとは [キャンセル] を繰り返しクリックして、起動中の RLogin はいったん閉じてしまって構いません。RLogin は、後でサーバへ入るときに使いますので、デスクトップの「rlogin_x64」フォルダと、その中にある「RLogin.exe」をごみ箱へ捨てないように注意してください。メモした公開鍵も無くさないようご注意ください。

【コラム】SSH の秘密鍵にパスフレーズは設定すべき？

秘密鍵に [パスフレーズ] を設定しておくと、鍵を使って SSH でサーバに入ろうとしたとき、「鍵を発動するにはパスフレーズを叫べ…！」という感じでパスフレーズを聞かれます。

つまり、もしあなたの秘密鍵が盗まれて誰かに勝手に使われそうになっても、パスフレーズを設定していれば鍵の悪用が防げます。スマホ本体が盗まれてしまっても、パスワードが分からなければロック画面が解除できず、勝手に使えないのと同じです。

これは「あなたは何を持っているのか」「あなたは何を知っているのか」「あなたは誰なのか」という複数の要素の中から、2つを用いることで認証の強度を高める「二要素認証」と呼ばれる考え方です。「あなたは秘密鍵を持っている」「あなたはパスフレーズを知っている」という2つの要素を組み合わせることで、単要素での認証よりも強度が高まります。ちなみに「あなたが誰なのか」は指紋認証や顔認証ですね。

ですが「パスワード認証じゃなくて鍵認証なのに、やっぱりパスフレーズが要るの…？」という具合に、初心者を混乱に陥れやすいので、本著では秘密鍵をパスフレーズなしで作って使います。

パスフレーズは「設定していれば絶対に安心！」というものではありませんが、上記の理由から、本来であれば設定した方がいいものです。後で「やっぱり設定しておこう」と思ったら、一度作成した秘密鍵に後からパスフレーズを設定することも可能です。

2.1.3 Mac でターミナルを準備する

Mac を使っている方は、最初から「ターミナル」(図 2.14) というソフトがインストールされていますのでそちらを利用しましょう。



▲図 2.14 最初からインストールされている「ターミナル」を使おう

ターミナルがどこにあるのか分からぬときは、Mac の画面で右上にある虫眼鏡のマークをクリックして、Spotlight で「ターミナル」と検索（図 2.15）すれば起動できます。



▲図 2.15 どこにあるのか分からなかつたら Spotlight で「ターミナル」と検索

2.1.4 Mac で SSH のキーペア（秘密鍵・公開鍵）を作成する

Mac の方は、ターミナルで次のコマンドを実行してください。⁵

```
$ ssh-keygen -f ~/Desktop/startSSLKey
```

すると次のように、パスフレーズの入力待ち状態になります。何も入力せずに、2回 Enter を押してください。

```
$ ssh-keygen -f ~/Desktop/startSSLKey
Generating public/private rsa key pair.
Enter passphrase (empty for no passphrase): ←何も入力せずに Enter
Enter same passphrase again: ←何も入力せずに Enter
```

次のように表示されたらキーペア（秘密鍵・公開鍵）の作成は完了です。

```
$ ssh-keygen -f ~/Desktop/startSSLKey
Generating public/private rsa key pair.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/mochikoAsTech/Desktop/startSSLKey.
Your public key has been saved in /home/mochikoAsTech/Desktop/startSSLKey.pub.
The key fingerprint is:
a2:52:43:dd:70:5d:a8:4f:77:47:ca:f9:69:79:14:48 mochikoAsTech@ghana
The key's randomart image is:
+--[ RSA 2048]----+
|       . .. ooE. |
|       . + o . . |
|       . . . . +. |
|       . . . = o |
|       o . So . . +o |
|       . o . . +o |
|       . . . . |
|       .           |
+-----+
```

ホームディレクトリに秘密鍵（startSSLKey）と、公開鍵（startSSLKey.pub）ができる

⁵ ssh-keygen コマンドは名前のとおり、SSH の鍵（key）を生成（generate）するコマンドです。-f オプションでは、生成する鍵のファイル名を指定しています。～（チルダ）はホームディレクトリを表しますので、-f ~/Desktop/startSSLKey は「/Users/<ユーザ名>/Desktop」のフォルダの中に「startSSLKey」という名前の鍵を作って、という意味です

あがっているはずです。cat（キャット）コマンド^{*6}で公開鍵を表示してみましょう。

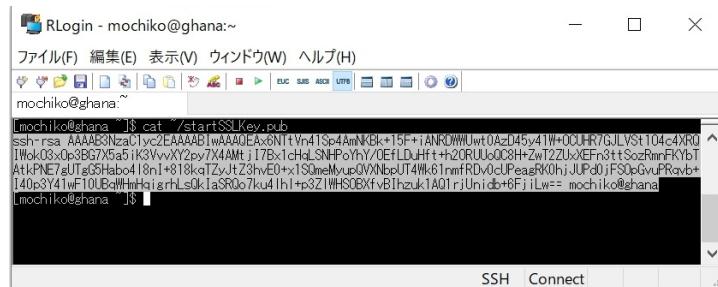
```
$ cat ~/startSSLKey.pub
ssh-rsa AAAAB3NzaC1yc2EAAQEAx0NtVn4TSpc4AmNKB+15f+iANR0WWUwt0AzD45y41W+00UHR7GJLVS1t04c4XRQ
Iwok03xOp3B67X5a1K3VyyXY2pY7X4AMfj17Bx1chLsNHPoYhY/0EfLDUhft+h20RUuOC8H+ZwT2ZuXEFn3ttSozRmnFKYbt
AtkPNE/gUlg5Habo418n1+818kgq1ZyJtZ3hvU0+x1SuMeMyupQVNnbU14Wk61nmfRDvJcUFeaghk0hjJUPd0)FSUpGvuRavb+
[40p3Y41wF10UBdIffhajgrhLs0k1aSR067ku4lh1+pSZIHSDOBxfvB1hzukTA01rjUnidb+6FjiLw== mochiko@ghana]
```

この後すぐに使いますので、表示された公開鍵（ssh-rsa から始まる文字列）をまるごとコピーして、メモ帳などにペーストしておきましょう。

以上で事前準備は完了です。お待たせしました。いよいよサーバを立てましょう。

【コラム】ターミナルでコピー＆ペーストするには？

ターミナルで表示されている内容をコピーしたいときは、コピーしたい部分をマウスで選択するだけです。（図 2.16）選択してから Ctrl+c を押す必要はありません。



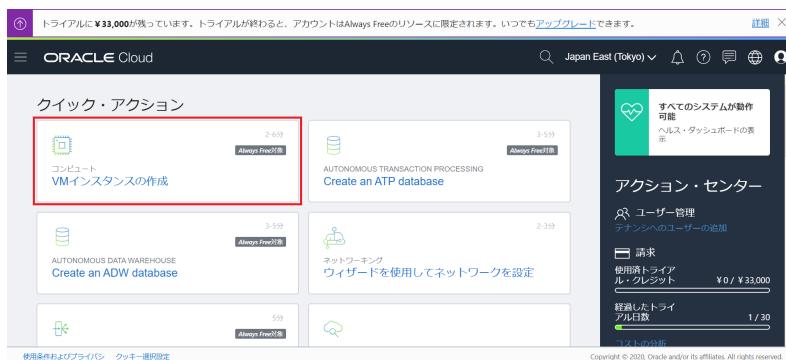
▲図 2.16 マウスで選択するだけでコピーできる

逆にコピーした内容をターミナルへペーストしたいときはターミナル上で**右クリックするだけ**です。ターミナルのソフトにもよりますが、基本的に Ctrl+p は使えないで注意してください。

^{*6} cat は猫ではなく「conCATenate files and print on the standard output」の略です

2.2 コンピュートでサーバを立てる

それでは下準備ができたので、Oracle Cloud のコンソールに戻ってサーバを立てましょう。コンソールにサインインしたら、[VM インスタンスの作成] をクリック（図 2.17）します。ちなみに Oracle Cloud にはデータベース管理やストレージなど、さまざまなサービスがありますが、クラウドサーバや物理サーバなどのサーバが立てられるサービスは「コンピュート」と呼ばれています。そして Oracle Cloud ではサーバのことを、**インスタンス**と呼びます。ここから先でインスタンスと書いてあつたら「サーバのことだな」と思ってください。



▲図 2.17 [VM インスタンスの作成] をクリック

[インスタンスの命名] に [startSSLInstance] と入力します。（図 2.18）その下の [オペレーティング・システムまたはイメージ・ソースを選択します] は、何も変更せずそのまま構いません。



▲図 2.18 [インスタンスの命名] に [startSSLInstance] と入力

2.2.1 OS は Oracle Linux 7.7 を使おう

パソコンには OS という基本ソフトが入っていて、Word や Excel、Chrome といったソフトはその OS の上で動いています。皆さんのパソコンにも「Windows 10」や「Mac OS X Lion」などの OS が入っていますよね。

そしてパソコンと同じようにサーバにも「Linux」や「Windows Server」といったサーバ用の OS があります。サーバを立てるときには Linux を選択することが多いのですが、この Linux の中にもさらに「RHEL (Red Hat Enterprise Linux)」や「CentOS」、「Ubuntu」などいろいろなディストリビューション（種類）があります。

本著では、OS はデフォルトの [Oracle Linux 7.7] を使用します。Oracle Linux なら Oracle Cloud のツールがあらかじめ入っていますので、**Oracle Linuxでサーバを立てるときはOSはOracle Linuxにすることをお勧めします**。Oracle Linux は Red Hat 系のディストリビューションですので、RHEL や CentOS のサーバを使ったことがある方なら違和感なく使えると思います。

2020 年 1 月時点で、Oracle Linux には次の 2 種類があります。

- Oracle Linux 6.10
- Oracle Linux 7.7

名前のとおり、Oracle Linux 6.10 は CentOS 6 と同じ RHEL6 系、Oracle Linux 7.7 は CentOS 7 と同じ RHEL7 系なので、使い勝手はほぼ同じです。

2.2.2 作っておいた SSH の公開鍵を設置しよう

さらに下に進んで [SSH キーの追加] は、[SSH キーの貼付け] を選択して、そこに先ほどメモしておいた公開鍵をペーストします。公開鍵は改行を含まず、先頭の「ssh-rsa」から末尾の「<ユーザ名>@<ホスト名>」のようなコメントまでで、まるごと 1 行です。(図 2.19)



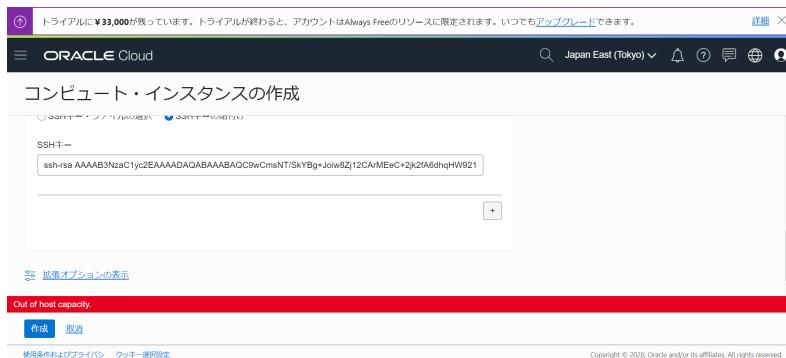
▲図 2.19 [SSH キーの貼付け] を選択してメモしておいた公開鍵をペースト

公開鍵をペーストしたら [作成] をクリックします。

2.2.3 <トラブル> "Out of host capacity."が起きたらどうすればいい？

さて、元気よく [作成] をクリックしたのに、真っ赤な [Out of host capacity.] が表示されてしまった…という方が一定数いらっしゃると思います。大丈夫、あなたは悪くありません。いま理由を説明するので落ち着いてください。「そんなの表示されなかったよ？」という方は、このコラムは読み飛ばして、この先の「サーバが起動するまで待とう」までジャンプしてください。

2.2 コンピュートでサーバを立てる



▲図 2.20 "Out of host capacity."と表示されて何も起きない！

"Out of host capacity."は、直訳すると「ホスト容量が不足しています」という意味ですが、ホストってなんでしょう？

あなたがいま Oracle Cloud で立てようとしたサーバは、家でいうと「一軒家」ではなく、マンションの 101 号室や 403 号室のような「各部屋」にあたります。このときマンションの建物をホストサーバ、各部屋をゲストサーバと呼びます。



▲図 2.21 マンションの建物をホストサーバ、各部屋をゲストサーバと呼ぶ

「ホストの容量が不足している」ということは…つまり、あなたが Oracle Cloud の無料マンションに入居しようとしたら、「ごめんね、無料マンションは大人気でいま空き部屋がないの」と断られてしまった、という状況なのです。

Oracle Cloud の Always Free は有効期限なしでずっと無料で使える、とても魅力的なサービスです。そのため Oracle Cloud 側も定期的に新築マンションを追加しているもの

の、定期的にリソース不足に陥ってはこういう状況になるようです。

この"Out of host capacity."が発生してしまった場合、次のどちらかが起きてホストの容量不足が解消しない限り、Always Free の枠でサーバは立てられません。

- 自分以外のユーザーがサーバを解約してリソースを開放する
- Oracle Cloud がリソースを増やす

ですが、Always Free とは別に、我々には 30 日間だけ有効な\$300 の無償クレジットが与えられています。たとえ無料マンションが満室でも、有料マンションなら空きがあります。30 日経ったら消えてしまう\$300 のお小遣いを握りしめたら、次の方法で有料マンションのお部屋を借りにいきましょう！

2.2.4 Always Free ではなく無償クレジットの枠でサーバを立てよう

次の手順は、"Out of host capacity."が表示された人だけ実施してください。

もともと選択していたのは[Always Free 対象]のマークが付いた[VM.Standard.E2.1.Micro(仮想マシン)] という種類のサーバでした。"Out of host capacity."を回避するため、少し上にスクロールして【シェイプ、ネットワークおよびストレージ・オプションの表示】をクリックしましょう。(図 2.22)



▲図 2.22 【シェイプ、ネットワークおよびストレージ・オプションの表示】をクリック

Oracle Cloud では、サーバスペックごとに「シェイプ」という区分があります。^{*7}インスタンスのシェイプを、いま選択されている [VM.Standard.E2.1.Micro] から変更した

^{*7} シェイプとはサーバスペックごとの区分のことです。AWS のインスタンスタイプと同じものだと思ってください

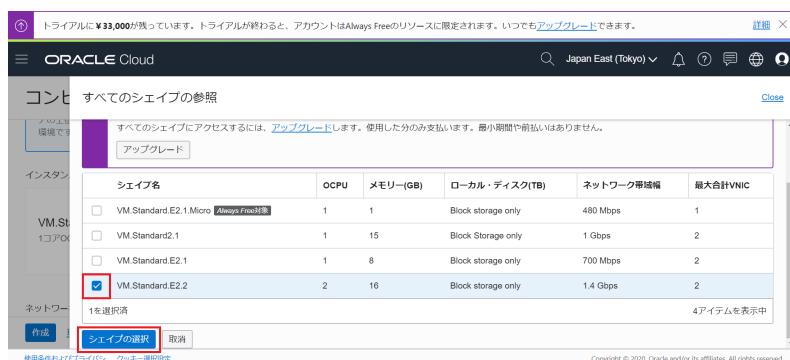
2.2 コンピュートでサーバを立てる

いので [シェイプの変更] をクリックしてください。



▲図 2.23 [シェイプの変更] をクリック

OCPU^{*8}が 2、メモリが 16GB の [VM.Standard.E2.2]^{*9}にチェックを入れて、[シェイプの選択] をクリックしましょう。



▲図 2.24 「VM.Standard.E2.2」に変更して [シェイプの選択] をクリック

それ以外は何も変更せずに、いちばん下の [作成] をクリックします。

*8 OCPU は Oracle Compute Units の略で、ごく簡単に言うと物理 CPU です。OCPU (物理 CPU) 1つは、vCPU (仮想 CPU) 2つに相当しますので、もし「AWS の EC2 で vCPU が 4 のサーバを使っている。同等スペックのサーバを用意してほしい」と頼まれたら、Oracle Cloud では OCPU が 2 のシェイプを選べば大丈夫です。単純に数字だけで比較して、OCPU が 4 のシェイプを選ぶと CPU のスペックがいままでの倍になってしまいますので注意してください

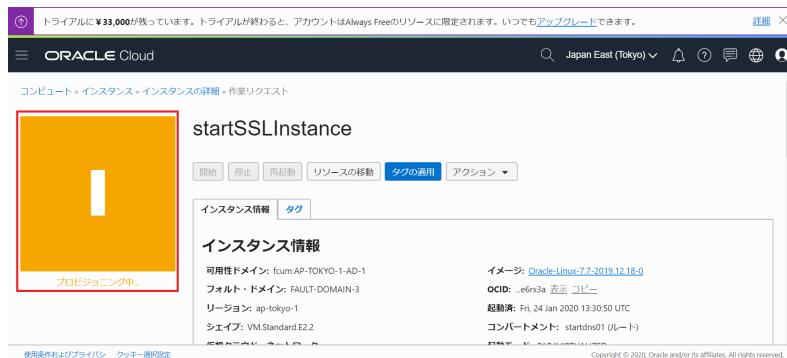
*9 シェイプの名前は、まず接頭辞が「VM」なら仮想サーバ (Virtual Machine)、「BM」なら物理サーバ (Bare Metal) を表しています。その後ろの単語は「Standard」(汎用) や「DenseIO」(高密度 IO) といった特徴、3番目の「E2」や「3」はシェイプの世代、最後の「2」や「8」は OCPU の数を表しています



▲図 2.25 [作成] をクリック

2.2.5 サーバが起動するまで待とう

オレンジ色で「[プロビジョニング中...]」と表示されたら、サーバが用意されるまでそのまま数分待ちましょう。



▲図 2.26 「[プロビジョニング中...]」と表示されたら数分待つ

サーバができあがると、表示が緑色の「[実行中]」に変わります。おめでとうございます！ これでサーバが立てられました！



▲図 2.27 「実行中」に変わった！ サーバが立てられた！

【コラム】Oracle Cloud のコンピュートの金額計算方法

ところで、いま立てた「VM.Standard.E2.2」をまるまる 1 ヶ月使ったら、一体いくら分になるのでしょうか？ うっかり \$300 を超えてしまわないか、ちょっと心配なので計算してみましょう。

コンピュートの価格表^{*10}を見てみると、[VM.Standard.E2.2] は [\$.03]^{*11}と書いてあります。これは [Pay as You Go (OCPU Per Hour)] と書いてあるとおり、1OCPU につき 1 時間あたりかかる金額です。^{*12}

「VM.Standard.E2.2」は OCPU が 2 なので、\$.03*2 で 1 時間あたり \$.06 かかることが分かります。1 ヶ月を 744 時間 (24 時間*31 日) として、\$.06*744 時間で \$44.64 です。

「VM.Standard.E2.2」を 1 台立てたくらいでは、\$300 の無償クレジットを使い切ることはないので安心しましょう。ちなみに Oracle Cloud では \$1 は 120 円換算^{*13}なので、日本円だと 5356.8 円ですね。

^{*12} <https://www.oracle.com/jp/cloud/compute/pricing.html>

^{*13} 2020 年 1 月時点の金額

^{*14} ちなみに AWS は、同スペックのサーバでもリージョンごとに価格が異なりますが、Oracle Cloud はどここのリージョンでも同一の価格です

^{*15} \$1 を 120 円で換算すると \$44.64*120 円で 5356.8 円です

【コラム】Oracle Cloud と AWS はどっちが安い？

Oracle Cloud は他のクラウドに比べて価格が安いのが特徴のひとつです。どれくらい安いのか、同じスペックのサーバで AWS と比較してみましょう。

例えば同スペックの VM.Standard.E2.1 (Oracle Cloud) と m5.large (AWS) を比較すると、Oracle Cloud の価格は AWS の 4 分の 1 以下です。(表 2.1)

▼表 2.1 Oracle Cloud と AWS の価格比較

	Oracle Cloud	AWS
インスタンスの種類	VM.Standard.E2.1	m5.large
CPU	OCPU:1 (vCPU:2相当)	vCPU:2
メモリ	8GB	8GB
1 時間あたり	\$0.03	\$0.124
月額	2678.4 円	11070.72 円

シェアトップを独走する AWS に対して、後発は勝つためにコスト面や性能面でそれぞれ大きなメリットを打ち出してきています。AWS が最適なのであれば AWS を選択すべきですが、「みんなが使っているから」というだけ理由で、あまり深く考えずに AWS を使っているのであれば、他のクラウドにも目を向けてみることを筆者はお勧めします。

2.2.6 接続先となるサーバの IP アドレス

無事にサーバが「実行中」になったら、接続先となるサーバの IP アドレスを確認してみましょう。

先ほど作成したインスタンス [startSSLInstance] の、[プライマリ VNIC 情報] (図 2.28) にある [パブリック IP アドレス] をメモ (表 2.2) してください。

2.2 コンピュートでサーバを立てる

The screenshot shows the Oracle Cloud Infrastructure (OCI) console. At the top, there's a banner with a purple circle containing a white question mark, followed by the text: 'トライアルに ¥32,719が残っています。トライアルが終わると、アカウントはAlways Freeのリソースに限定されます。いつでも[アップグレード](#)できます。' Below the banner is the Oracle Cloud logo. The main content area has a green header bar with the text '実行中' (Running). The main body contains two sections: 'インスタンス情報' (Instance Information) and 'プライマリ VNIC 情報' (Primary VNIC Information). The 'Primary VNIC Information' section includes fields for Private IP (10.0.2), Public IP (140.238.33.51), Internal FQDN (startstestinstance...), and Subnet (Public - Subnet). There are also links for Oracle Linux 7.7-20191218-0 and startstest01 (ルート). The bottom of the page includes a footer with copyright information: 'Copyright © 2020, Oracle and/or its affiliates. All rights reserved.'

▲図 2.28 [プライマリ VNIC 情報] の [パブリック IP アドレス] をメモしておこう

▼表 2.2 インスタンスの [パブリック IP アドレス]

例	パブリック IP アドレス
	140.238.33.51

それではメモした IP アドレスを使ってサーバに入ってみましょう。

第3章

ウェブサーバの設定をしよう

この章ではウェブサーバの設定を行ないます。

3.1 サーバに SSH でログインしよう

では立てたばかりのサーバに、SSH でログインしてみましょう。

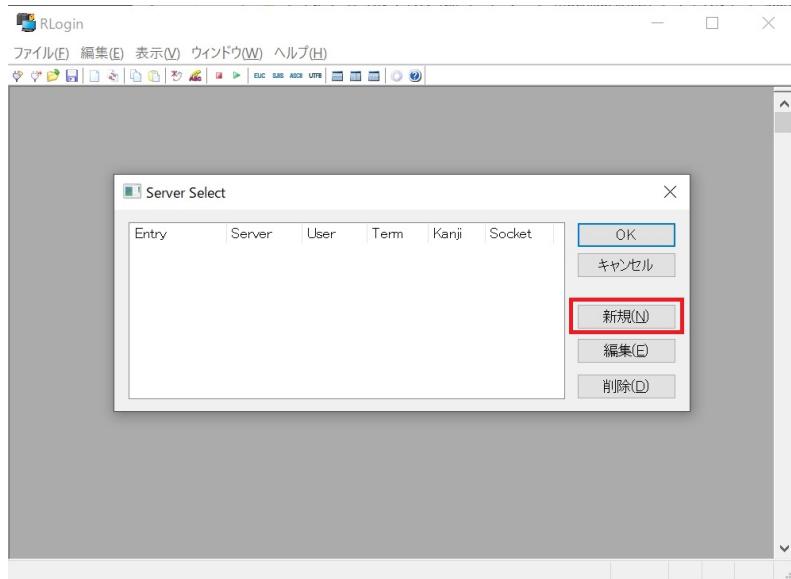
3.1.1 Windows の RLogin を使ってサーバに入ってみよう

Windows のパソコンを使っている方は、デスクトップの [rlogin_x64] というフォルダの中にある [RLogin.exe]（図 3.1）をダブルクリックして RLogin を起動（図 3.2）してください。起動したら [新規] をクリックします。



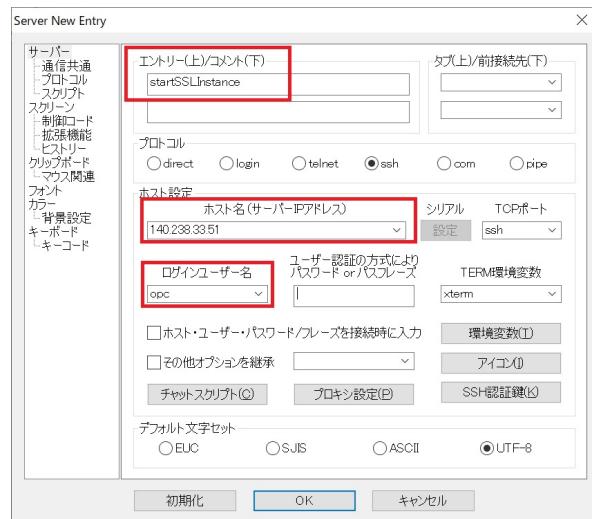
▲図 3.1 RLogin.exe をダブルクリック

3.1 サーバに SSH でログインしよう



▲図 3.2 RLogin が起動したら [新規] をクリック

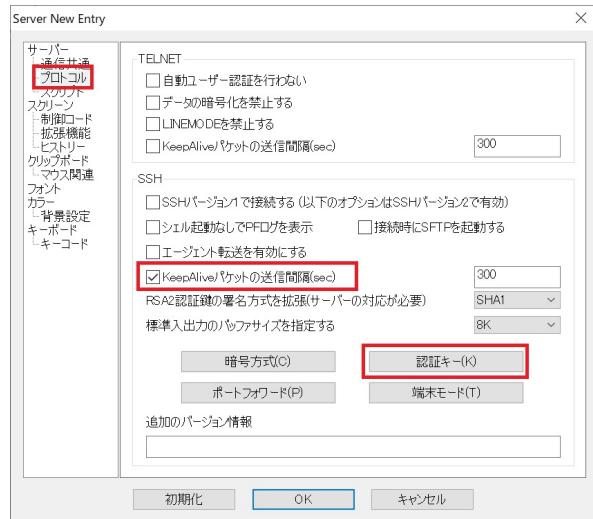
初めに [エントリー (上) /コメント (下)] の上に [startSSLInstance] と入力します。続いて [ホスト名 (サーバー IP アドレス)] に先ほどメモした [パブリック IP アドレス] を入力 (図 3.3) します。[ログインユーザー名] には [opc] と入力してください。opc というのは Oracle Linux のインスタンスを作成すると、最初から存在しているデフォルトユーザです。



▲図 3.3 [ホスト名 (サーバー IP アドレス)] と [ログインユーザー名] を入力

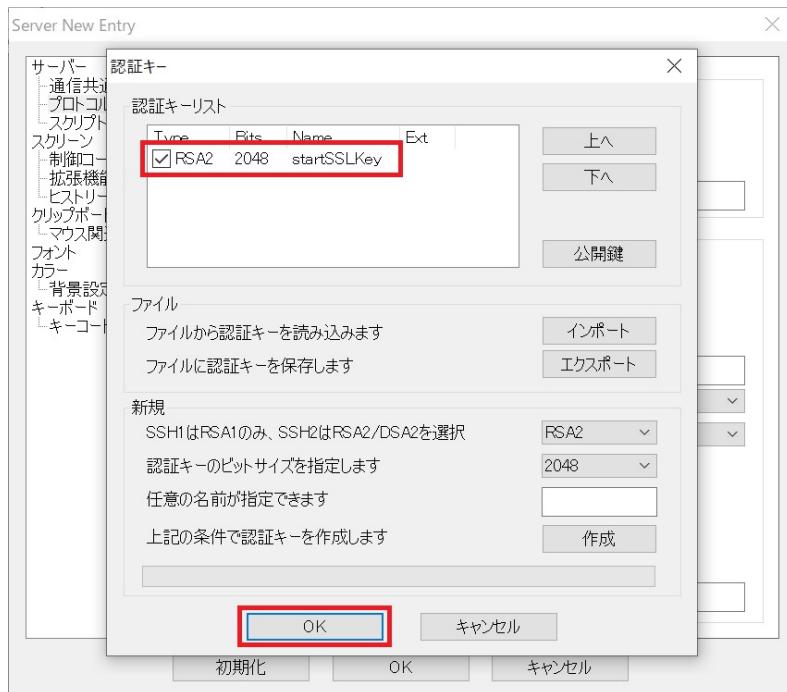
次に左メニューで [プロトコル] を選択（図 3.4）したら、[KeepAlive パケットの送信間隔 (sec)] にチェックを入れておきます。これを設定しておくとターミナルをしばらく放っておいても接続が勝手に切れません。続いて [認証キー] をクリックします。

3.1 サーバに SSH でログインしよう



▲図 3.4 [KeepAlive パケットの送信間隔 (sec)] にチェックを入れて [認証キー] をクリック

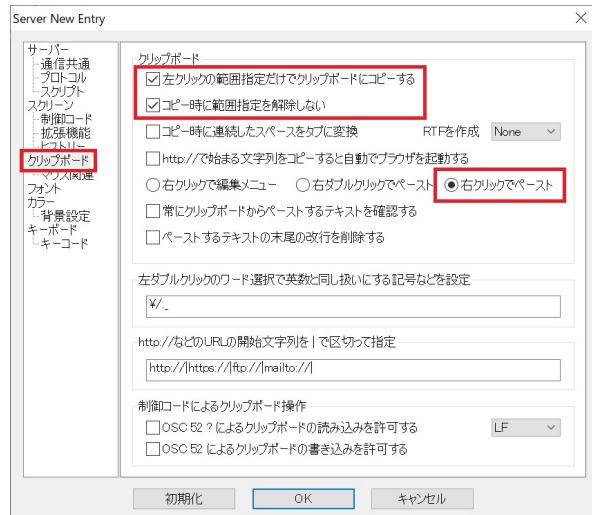
[認証キー] リストで [startSSLKey] にチェックが入っていることを確認（図 3.5）します。これは「ログインするときにこの鍵を使います」というリストです。チェックが入っていたら [OK] をクリックして閉じて構いません。



▲図 3.5 [startSSLKey] にチェックが入っていることを確認

続いて左メニューで「クリップボード」を選択（図 3.6）したら、「左クリックの範囲指定だけでクリップボードにコピーする」と「コピー時に範囲指定を解除しない」にチェックを入れて「右クリックでペースト」を選択します。

3.1 サーバに SSH でログインしよう



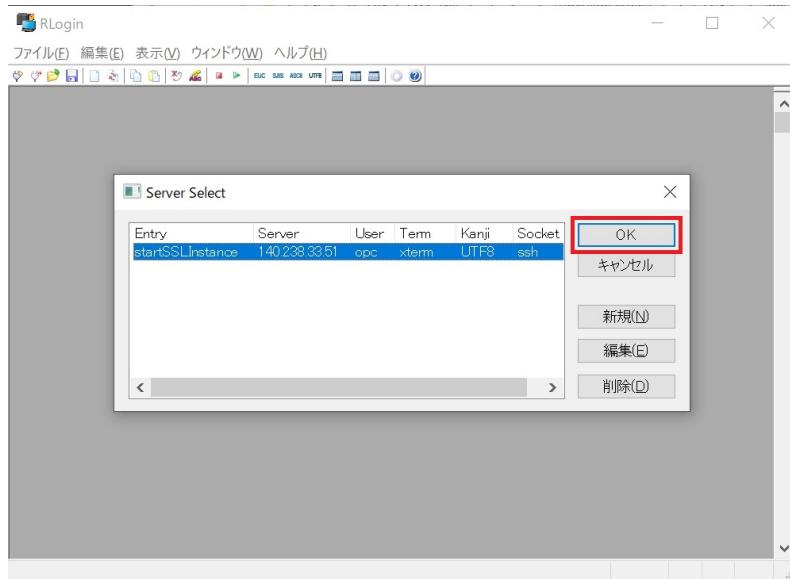
▲図 3.6 右クリックや左クリックの設定

次に左メニューで「フォント」を選択（図 3.7）したら、文字セットを「UTF-8」に変更します。すべて設定できたら「OK」をクリックしてください。



▲図 3.7 文字セットを「UTF-8」に変更

設定が保存できたら「OK」をクリック（図 3.8）してください。



▲図 3.8 設定が保存できたら「OK」をクリック

すると初回のみ、この「公開鍵の確認」が表示（図 3.9）されます。これは「初めて入るサーバだけど信頼していいですか？本当に接続しますか？」と聞かれているので、「接続する」をクリックしてください。サーバにはそれぞれフィンガープリントという固有の指紋があるため、下部の「この公開鍵を信頼するリストに保存する」にチェックが入っていれば RLogin が覚えていてくれて、次回以降は「これは前に信頼していいって言われたサーバだ！」と判断してそのまま接続させてくれます。



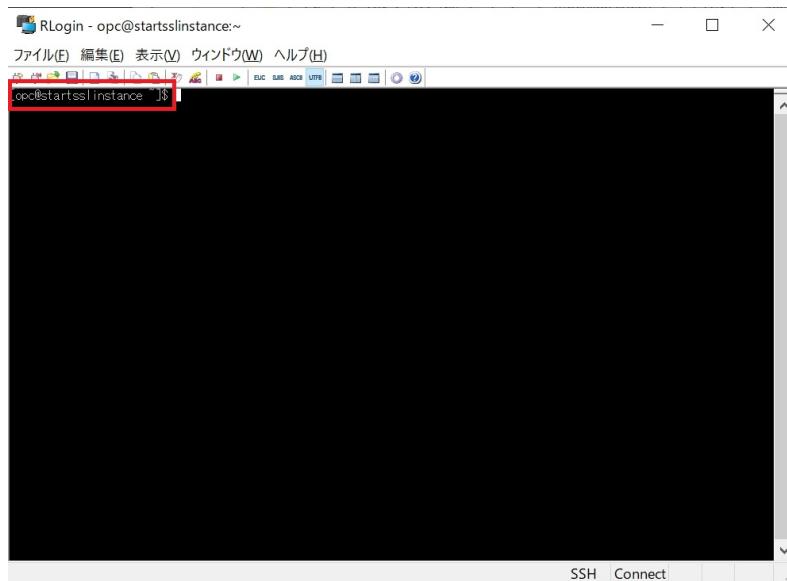
▲図 3.9 「公開鍵の確認」が表示されたら「接続する」をクリック

続いて「信頼するホスト鍵のリストを更新しますか？」と聞かれたら「はい」をクリック（図 3.10）してください。



▲図 3.10 「信頼するホスト鍵のリストを更新しますか？」と表示されたら「はい」をクリック

「opc@startsslinstance」と表示（図 3.11）されたら無事サーバに入っています。SSHでのログイン成功、おめでとうございます！



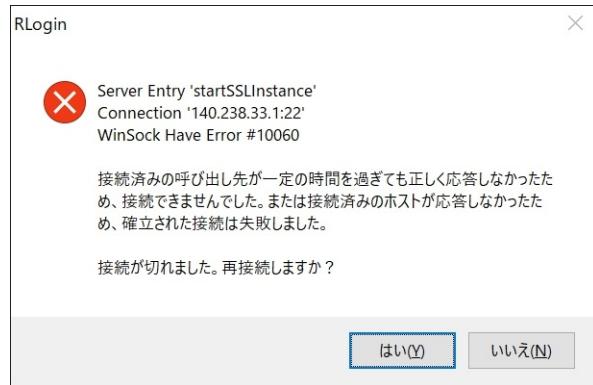
▲図 3.11 「opc@startsslinstance」と表示されたら成功！

もし「opc@startsslinstance」と表示されず、代わりに「SSH2 User Auth Failure "publickey,ssapi-keyex,ssapi-with-mic" Status=1004 Send Disconnect Message... ssapi-with-mic」というようなエラーメッセージが表示（図 3.12）されてしまったら、これは「鍵がない人は入れないよ！」とお断りされている状態です。【認証キー】リストで【startSSLKey】にチェックが入っていないものと思われますので【認証キー】の設定を確認してみてください。



▲図 3.12 このエラーが表示されたら【認証キー】を確認しよう

「接続済みの呼び出し先が一定の時間を過ぎても正しく応答しなかったため、接続できませんでした。」というエラーメッセージが表示（図 3.13）されてしまった場合は、「ホスト名（サーバー IP アドレス）」に書いた「パブリック IP アドレス」が間違っているものと思われます。「ホスト名（サーバー IP アドレス）」の IP アドレスを確認してみてください。



▲図 3.13 このエラーが表示されたら「ホスト名（サーバー IP アドレス）」の IP アドレスを確認しよう

3.1.2 Mac のターミナルを使ってサーバに入ってみよう

Mac を使っている方は、ターミナル（図 3.14）を起動してください。



▲図 3.14 最初からインストールされている「ターミナル」を使おう

ターミナルがどこにあるのか分からぬときは、Mac の画面で右上にある虫眼鏡のマー

クをクリックして、Spotlight で「ターミナル」と検索（図 3.15）すれば起動できます。



▲図 3.15 どこにあるのか分からなかったら Spotlight で「ターミナル」と検索

そして開いたターミナルで次の文字を入力して Return キーを押します。これはサーバに入るときに使う鍵をオーナー以外が使えないよう、chmod というコマンドで読み書き権限を厳しくしています。この作業は最初の 1 回だけで構いません。もし「startSSLKey」を保存した場所がデスクトップ以外の場合は適宜書き換えてください。

```
$ chmod 600 ~/Desktop/startSSLKey
```

続いてターミナルで次の文字を入力したら再び Return キーを押します。「パブリック IP アドレス」の部分は先ほどメモした「パブリック IP アドレス」に書き換えてください。-i オプションは「サーバにはこの鍵を使って入ります」という意味ですので、「startSSLKey」を保存した場所がデスクトップ以外だった場合はこちらも適宜書き換えてください。

```
$ ssh opc@パブリック IP アドレス -i ~/Desktop/startSSLKey
```

初回のみ次のようなメッセージが表示されますが、これは「初めて入るサーバだけど

信頼していいですか？本当に接続しますか？」と聞かれていますので、「yes」と打ってReturnキーを押してください。するとMacはちゃんとこのサーバのことを覚えてくれて、次回以降は「これは前に信頼していいって言われたサーバだ！」と判断してそのまま接続させてくれます。

```
Are you sure you want to continue connecting (yes/no)?
```

「opc@startsslinstance」と表示されたら無事サーバに入っています。おめでとうございます！

今後はいまやったのと同じやり方をそのまま繰り返せばサーバにログインできます。

3.2 ターミナルでサーバを操作・設定してみよう

ようやくサーバに入れたので、ここからはターミナルの基本的な操作を試してみましょう。

3.2.1 プロンプトとは？

では黒い画面で何回かEnterキー（あるいはReturnキー）を押してみましょう。（図3.16）普通に改行されますよね。



▲図3.16 Enterキーを押すと改行されて、プロンプトが常に表示されている

このとき左側にずっと出ている次のような表示は「プロンプト」といって、ログインしているユーザ名やサーバの名前などが表示されています。

```
[opc@startsslinstance ~]$
```

プロンプトを見るといまは「opc」という一般ユーザであることが分かります。これからサーバに色々な設定をしたいのですが、一般ユーザだと権限がないので「root」という全権限をもったユーザになりましょう。

```
$ sudo su -
```

と書いて Enter キーを押すと root になれます。(図 3.17) 「\$」はプロンプトを表していますので入力しないでください。root になれたらまた何回か Enter キーを押して改行してみましょう。



▲図 3.17 sudo su -を書いて Enter キーを押すと root になれる

いちばん左側に出ているプロンプトが次のように変化しましたか？

```
[root@startsslinstance ~]#
```

ユーザ名が「opc」から「root」に変わりました。それからいちばん右の部分も「\$」から「#」に変わっています。プロンプトは**一般ユーザだと「\$」で全権を持っているrootだと「#」**という表示になります。今後は「このコマンドを root で実行してください」のように実行ユーザを詳しく書くことはしませんので、例として書いてある部分のプロンプトが「\$」だったら opc のような一般ユーザで実行、「#」だったら root で実行するんだ、と思ってください。例に「\$」や「#」が書いてあってもターミナルで「\$」や「#」を自分で入力する必要はありません。

3.2.2 コマンドは失敗したときだけエラーを吐く

前述の「sudo su -」のようなものを「コマンド」と呼びます。コマンドとはサーバに対して「あれをして」「これをして」と頼む命令のようなものです。

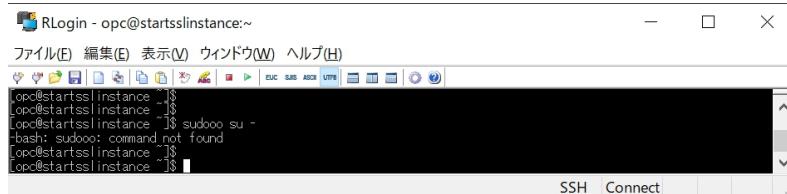
サーバでコマンドを打った場合、基本的に上手くいったときは何も言わないので失敗した

ときだけエラーを吐きます。 ですのでコマンドを打った時に何も表示されなくとも不安にならなくて大丈夫です。

先ほどの「私を root にして！」という命令である「sudo su -」も、上手くいってちゃんと root になれたのでメッセージは一切出ていないですよね。これが「sudo」を打ち間違えて、こんな風に実行するとどうなるでしょう？

```
$ sudooo su -
```

「sudooo なんてコマンドは見つからなかったよ」というエラーメッセージ（図 3.18）が表示されました。



▲図 3.18 「sudooo: command not found」というエラーが表示された

このように何か失敗したときだけエラーが出ます。英語でエラーが出るとそれだけでパニックになってしまいますが、落ち着いてゆっくり読めば「sudooo: command not found…ああ、sudooo っていうコマンドが見つかりませんでした、って書いてある」と判読できると思います。エラーが出たら声に出してゆっくり読んでみましょう。

3.2.3 ターミナルを閉じたいとき

もう今日の勉強は終わり！ サーバとの接続を切ってターミナルを閉じたい、というときは exit（イグジット）というコマンドを叩きます。

```
# exit
```

root になっているときに exit を叩くと opc に戻れます。そして opc で再び exit を叩くと、サーバの接続を切ってターミナルを閉じることができます。

```
$ exit
```

exit をせずに右上の赤い×を押してウィンドウを閉じるのは、電話を切るときに通話オフのボタンを押さずに電話線を引っこ抜くような乱暴な切り方なのでお勧めしません。

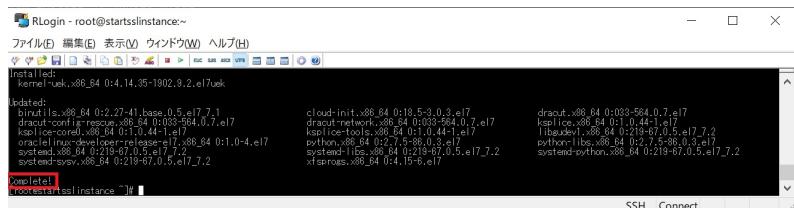
3.3 NGINX をインストールしよう

それでは必要なミドルウェアをインストールしていきましょう。最初に root になっておいてください。インストールするときは yum (ヤム) というコマンドを使います。

```
$ sudo su -
```

先ずは yum で色々アップデートしておきましょう。Windows アップデートみたいなものです。画面にたくさん文字が流れて、少し時間がかかりますが、最後に [Complete!] と表示されたら問題なく完了しています。(図 3.19) ちなみに-y オプションは YES を意味するオプションです。-y オプションをつけないで実行すると「これとこれを更新するけどいい？ ダウンロードサイズとインストールサイズはこれくらいだよ」という確認が表示されて、y と入力して Enter キーを押さないと更新されません。

```
# yum update -y
```



▲図 3.19 最後に [Complete!] と表示されたらアップデートは完了

続いて NGINX^{*1}を入れます。2020 年 1 月現在の安定バージョン^{*2}である 1.16 系をイ

^{*1} NGINX と書いて「えんじんえっくす」と読みます。ウェブサーバのミドルウェアの中で NGINX は順調にシェアを伸ばし、2019 年 4 月にとうとう Apache を抜いてシェア 1 位になりました。https://news.mynavi.jp/article/20190424-813722/

^{*2} サーバに入っている NGINX のバージョンがいくつなのか？ という情報は大切です。今後、あなたが

第3章 ウェブサーバの設定をしよう

インストールしたいので、yum のリポジトリ（どこから NGINX をダウンロードしてくるか）に NGINX 公式を追加しましょう。

```
# rpm -ivh http://nginx.org/packages/centos/7/noarch/RPMS/nginx-release-centos-7-0.el7.ngx-1.16.1-1.el7.noarch.rpm
# cat /etc/yum.repos.d/nginx.repo

↓以下が表示されればOK
# nginx.repo

[nginx]
name=nginx repo
baseurl=http://nginx.org/packages/centos/7/$basearch/
gpgcheck=0
enabled=1
```

yum で NGINX をインストールします。

```
# yum install -y nginx
```

[Complete!] と表示されたらインストール完了です。バージョン情報を表示することで、ちゃんとインストールされたか確認してみましょう。

```
# nginx -v
nginx version: nginx/1.16.1 ←バージョン情報が表示されればインストールできている
```

ちょっと分かりにくいかも知れませんが、パソコンに Microsoft Excel をインストールしたら「表計算というサービスが提供できるパソコン」になるのと同じで、サーバにこの NGINX をインストールすると「リクエストに対してウェブページを返すサービスが提供できるサーバ」、つまりウェブサーバになります。今回は NGINX を入れましたが、ウェブサーバのミドルウェアは他にも Apache をはじめとして色々な種類があります。

インストールが終わったので、サーバを再起動した場合も NGINX が自動で立ち上がりてくるよう、自動起動の設定もオンにしておきましょう。systemctl コマンドで、NGINX の自動起動を有効 (enable) にします。

NGINX の設定ファイルを書こうと思って調べたとき、例えば 1.12 系の設定方法を参考にしてしまうと、1.16 系の NGINX の環境では上手く動かない可能性があります。

```
# systemctl enable nginx
# systemctl is-enabled nginx
enabled ←有効になったことを確認
```

3.4 Firewalld で HTTP と HTTPS を許可しよう

続いてサーバの中で動いているファイアウォールの設定を変更します。まずは現状、何がファイアウォールを通れるようになっているのか確認してみましょう。

```
# firewall-cmd --list-services
dhcpv6-client ssh
```

dhcpv6-client と ssh は通つていいけれど、それ以外は誰であろうと通さないぞ！ という設定になっています。このままではブラウザでサイトを見ようとしても、ウェブページを返してくれるはずの NGINX までリクエストが届きません。そこで次のように、許可対象に http と https を追加して、変更が反映されるよう再読み込みします。それぞれ [success] と表示されたら成功しています。

```
# firewall-cmd --add-service=http --permanent
success

# firewall-cmd --add-service=https --permanent
success

# firewall-cmd --reload
success
```

これでファイアウォールの設定が変更できたので、もう一度、誰がファイアウォールを通してもらえるのか確認してみましょう。http と https が追加されていれば問題ありません。

```
# firewall-cmd --list-services
dhcpv6-client http https ssh
```

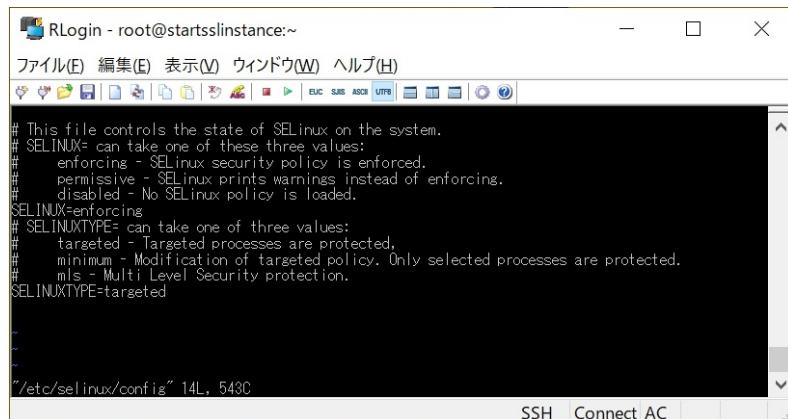
3.5 SELinux を無効にしておこう

SELinux を無効化 (disabled) します。^{*3}

```
# getenforce
Enforcing ←起動直後は有効になっている

# vi /etc/selinux/config
```

vi (ブイアイ) はテキストファイルを編集するためのコマンドです。vi コマンドでファイルを開くと、最初は次のような「閲覧モード」の画面（図 3.20）が表示されます。閲覧モードは「見るだけ」なので編集ができません。



▲図 3.20 vi コマンドでファイルを開いた

この状態で i (アイ) を押すと「編集モード」^{*4}に変わります。（図 3.21）左下に「-- INSERT --」と表示されていたら「編集モード」です。

^{*3} SELinux は Security-Enhanced Linux の略で、セキュリティポリシーを設定することで、Linux 上のアプリケーションやプロセス、そしてファイルへ細かくアクセス制御できる機能です。設定は間違っていないはずなのに、なぜかアクセスできない…というときは大概 SELinux が原因です。本来はセキュリティポリシーをきちんと設定するのが筋なのかも知れませんが、「SELinux を無効にしないでちゃんと使ってるところってあるのかな？」と同僚に聞いたら、「俺、NASA しか知らない」という返事をもらったあの日から、筆者は SELinux を無効にすることをためらわなくなりました

^{*4} ここでは初心者の方でも直感的に分かるよう「閲覧モード」「編集モード」と呼んでいますが、正しくは「ノーマルモード」「インサートモード」です。

```
# This file controls the state of SELinux on the system.
# SELINUX can take one of these three values:
#       enforcing - SELinux security policy is enforced.
#       permissive - SELinux prints warnings instead of enforcing.
#       disabled - No SELinux policy is loaded.
SELINUX=enforcing
# SELINUXTYPE can take one of three values:
#       targeted - Targeted processes are protected,
#       minimum - Modification of targeted policy. Only selected processes are protected.
#       mls - Multi Level Security protection.
SELINUXTYPE=targeted

-- INSERT --
```

SSH Connect AC

▲図 3.21 i (アイ) を押すと「-- INSERT --」と表示される「編集モード」になった

「編集モード」になるとファイルが編集できるようになります。それでは「SELINUX=enforcing」を「SELINUX=disabled」（図 3.22）に書き換えてください。

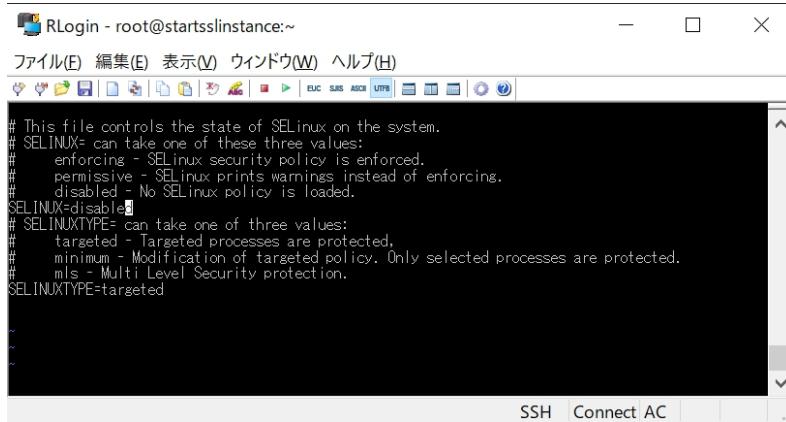
```
# This file controls the state of SELinux on the system.
# SELINUX can take one of these three values:
#       enforcing - SELinux security policy is enforced.
#       permissive - SELinux prints warnings instead of enforcing.
#       disabled - No SELinux policy is loaded.
SELINUX=disabled
# SELINUXTYPE can take one of three values:
#       targeted - Targeted processes are protected,
#       minimum - Modification of targeted policy. Only selected processes are protected.
#       mls - Multi Level Security protection.
SELINUXTYPE=targeted

-- INSERT --
```

SSH Connect AC

▲図 3.22 「SELINUX=enforcing」を「SELINUX=disabled」に書き換える

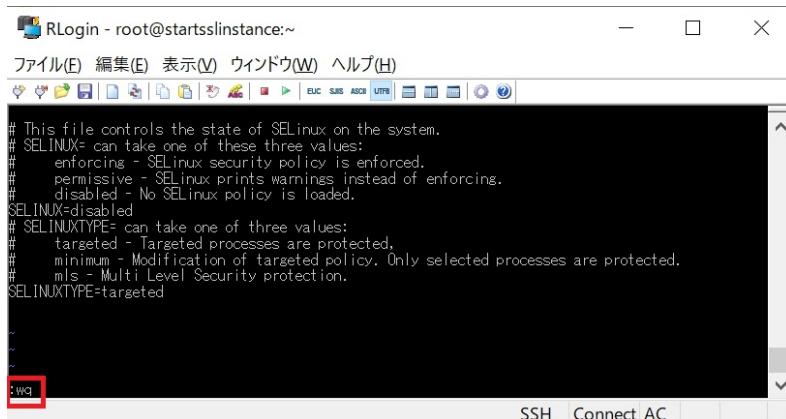
「編集モード」のままだと保存ができないので書き終わったら ESC キーを押します。すると左下の「-- INSERT --」が消えて再び「閲覧モード」になります。（図 3.23）



```
# This file controls the state of SELinux on the system.
# SELINUX can take one of these three values:
#       enforcing - SELinux security policy is enforced.
#       permissive - SELinux prints warnings instead of enforcing.
#       disabled - No SELinux policy is loaded.
SELINUX=disabled
# SELINUXTYPE can take one of three values:
#       targeted - Targeted processes are protected,
#       minimum - Modification of targeted policy. Only selected processes are protected.
#       mls - Multi Level Security protection.
SELINUXTYPE=targeted
```

▲図 3.23 ESC を押すと左下の「-- INSERT --」が消えて再び「閲覧モード」になる

「閲覧モード」に戻ったら「:wq」⁵と入力して Enter キーを押せば変更が保存されます。(図 3.24)



```
# This file controls the state of SELinux on the system.
# SELINUX can take one of these three values:
#       enforcing - SELinux security policy is enforced.
#       permissive - SELinux prints warnings instead of enforcing.
#       disabled - No SELinux policy is loaded.
SELINUX=disabled
# SELINUXTYPE can take one of three values:
#       targeted - Targeted processes are protected,
#       minimum - Modification of targeted policy. Only selected processes are protected.
#       mls - Multi Level Security protection.
SELINUXTYPE=targeted
```

▲図 3.24 「:wq」と入力して Enter キーを押せば保存される

色々やっているうちになんだか訳が分からなくなってしまって「いまの全部なかったことにしたい！ 取り合えず vi からいったん抜けたい！」と思ったときは、ESC キーを押して「:q!」⁶と入力して Enter キーを押すと変更を保存せずに抜けることができます。

*5 書き込んで (write)、抜ける (quit) という命令なので wq です。

*6 保存せずに強制終了 (quit!) という命令なので q! です。

編集できたら cat (キャット) コマンドでファイルの中身を確認してみましょう。

```
# cat /etc/selinux/config  
  
# This file controls the state of SELinux on the system.  
# SELINUX= can take one of these three values:  
#       enforcing - SELinux security policy is enforced.  
#       permissive - SELinux prints warnings instead of enforcing.  
#       disabled - No SELinux policy is loaded.  
SELINUX=disabled  
# SELINUXTYPE= can take one of three values:  
#       targeted - Targeted processes are protected,  
#       minimum - Modification of targeted policy. Only selected processes are protected.  
#       mls - Multi Level Security protection.  
SELINUXTYPE=targeted
```

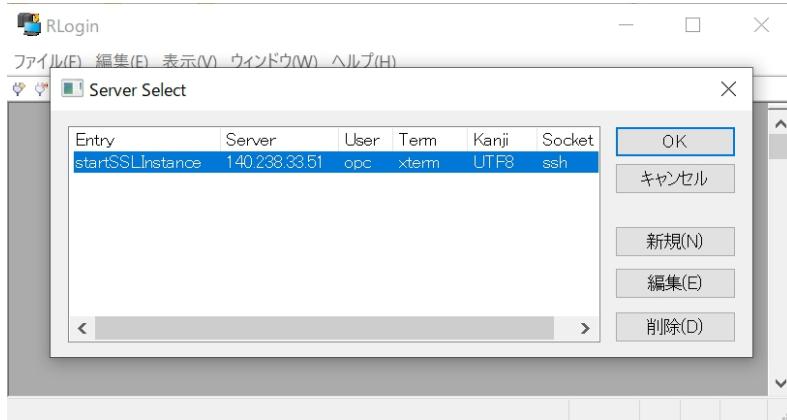
以上で SELinux を無効化する設定は終わりです。

3.6 OS を再起動してみよう

変更した設定を反映させるため reboot (リブート) コマンドでサーバを再起動しておきましょう。

```
# reboot
```

SSH の接続も切れてしまいますが、割とすぐに再起動しますので再度 RLogin やターミナルで接続（図 3.25）してみてください。今度はさっさと同じ設定でそのまま接続できるはずです。



▲図 3.25 さっそく同じ設定で接続してみよう

以上で「サーバを立てる」という作業はおしまいです。

3.6.1 ターミナルはなんのためにある？

ターミナルで yum や vi を叩いてサーバの設定を色々してきましたが、ここで「結局、ターミナルって何なの？」という振り返りをしておきましょう。

ターミナルはサーバを操作するための画面です。

皆さんのがパソコン使うときはモニタに表示された画面を見ながらキーボードとマウスを使って「フォルダを開いて先週作ったWordファイルを探す」とか「Wordファイルを開いて今週の報告書を書く」というような操作をするとと思います。フォルダを開くときは「ダブルクリック」をして、書いた内容を保存するときは「上書き保存する」ボタンを押しますよね。

サーバも同じです。サーバを使うときは「ターミナル」という画面を開いて操作します。ディレクトリ^{*7}を開いて移動するときはダブルクリックの代わりに cd^{*8}というコマンドを叩いて移動しますし、ディレクトリの中を見るときもダブルクリックでフォルダを開く代わりに ls コマンドを叩いて見ます。

皆さんのがいま使っているWindowsやMacといった「パソコン」だったらマウスやキーボードを使ってアイコンやボタンを見ながら操作できますが、サーバは基本的にこの真っ黒な「ターミナル」で文字を打って操作します。パソコンのときはダブルクリックやボタ

*7 Linuxではフォルダのことをディレクトリと呼びます。

*8 change directory の略。

ンを押す、という形で伝えていた命令がすべてコマンドに置き換わっていると思ってください。

パソコンもないのにマウスやキーボードだけあっても意味が無いように、ターミナルもそれ単体では何もできません。操作対象であるサーバがあって初めて役に立つ道具なのです。

ちなみにターミナルは背景の色も文字の色も好きに変えられます。どうしても「黒い画面怖い！」という感覚が抜けない人は、ピンクとかオレンジとか好きな色にしてみましょう。^{*9}

まとめるとターミナルとはサーバを操作するための画面で、操作するときにはコマンドという命令を使います。

3.7 なぜかサイトが見られない

ウェブサーバも立てたし、SELinuxはとめたし、サーバの中のファイアウォールに穴も空けました。これで準備完了！ サーバを立てたときにメモした「パブリック IP アドレス」を、ブラウザで開いてみました。するとしばらくぐるぐるした後で、「接続がタイムアウトしました」と表示されてしまいました。（図 3.26）



▲図 3.26 なぜか HTTP でサイトが表示されない…

403 や 404 や、あるいは 502 などのステータスコードも返ってきていないので、そもそも

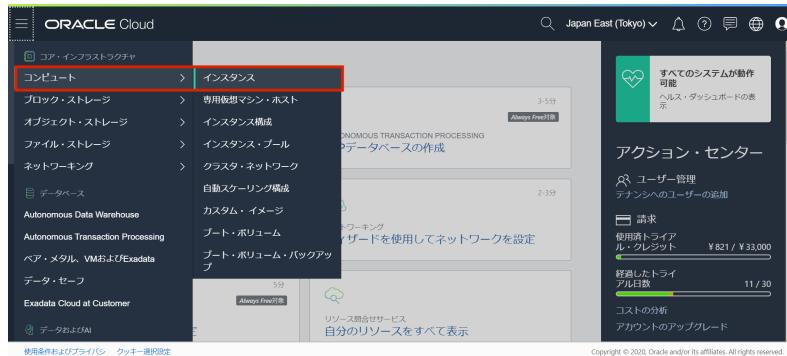
^{*9} Mac のターミナルはそもそも黒じゃなくて白ですね。

も NGINX までたどり着けていないようです。

3.7.1 サーバの手前にあるファイアウォールにも穴を空けよう

先ほどサーバの中にあるファイアウォールの設定を変更して、HTTP と HTTPS が通れるようにしましたが、実はサーバの中だけでなく、サーバの外にももう 1 つファイアウォールがいます。サイトが表示されなかったのは、「ウェブページを見せて！」というリクエストが、サーバの手前のファイアウォールで阻まれていたためなのです。サーバの手前にあるファイアウォールにも穴を空けて、HTTP と HTTPS がサーバまでたどり着けるようにしましょう。

再び Oracle Cloud のコンソールに戻って、左上の [ハンバーガーメニュー] から [コンピュート] の [インスタンス] を開きます。(図 3.27)



▲図 3.27 [ハンバーガーメニュー] から [コンピュート] の [インスタンス] を開く

インスタンスの一覧が表示されるので [startSSLInstance] をクリックします。(図 3.28)

3.7 なぜかサイトが見られない

The screenshot shows the Oracle Cloud Compute Instances list. A table displays the instance details for 'startdns01 (ルート)コンパートメント内のインスタンス'. One row is highlighted with a red border, showing the OCID as 'e6nx3a' and the name as 'startSSLInstance'. The status is '実行中' (Running). The table includes columns for Region, Shape, Availability Domain, and Last Refreshed.

▲図 3.28 [startSSLInstance] をクリック

[パブリック・サブネット] をクリックします。(図 3.29)

The screenshot shows the Oracle Cloud Instance Details page for 'startdns01 (ルート)'. The 'Instances Information' tab is selected. Under the 'Networking' section, the 'Public Network' tab is active. It shows the public IP address '140.238.33.51' and the internal FQDN 'startsslinstance'. The 'Subnet' field is highlighted with a red border, showing 'パブリック・サブネット'.

▲図 3.29 [パブリック・サブネット] をクリック

もう一度、[パブリック・サブネット] をクリックします。(図 3.30)

第3章 ウェブサーバの設定をしよう

The screenshot shows the Oracle Cloud Infrastructure console. The top navigation bar includes the Oracle Cloud logo, search bar, and region selection (Japan East (Tokyo)). Below the navigation is a green banner with the text '使用可能' (Available). The main content area has a title 'startdns01 (ルート) コンバートメント内のサブネット'. A table titled 'サブネットの作成' lists one item: 'パブリック・サブネット' (Public Subnet), which is marked as '使用可能' (Available). The table includes columns for Name, Status, CIDR Block, Subnet Access, and Creation Date.

▲図 3.30 もう一度、[パブリック・サブネット] をクリック

[セキュリティ・リスト] の中にある [Default Security List for VirtualCloudNetwork～] をクリックします。(図 3.31) このセキュリティ・リストが、サーバの手前にいるファイアウォールです。

The screenshot shows the Oracle Cloud Infrastructure console. The top navigation bar includes the Oracle Cloud logo, search bar, and region selection (Japan East (Tokyo)). Below the navigation is a green banner with the text '使用可能' (Available). The main content area has a title 'セキュリティ・リスト'. A table titled 'セキュリティ・リストの追加' lists one item: 'Default Security List for VirtualCloudNetwork-20200120-2319', which is marked as '使用可能' (Available). The table includes columns for Name, Status, Conversion Rule, and Creation Date.

▲図 3.31 [Default Security List for VirtualCloudNetwork～] をクリック

[イングレス・ルール] で、[イングレス・ルールの追加] をクリックします。(図 3.32)

3.7 なぜかサイトが見られない

The screenshot shows the Oracle Cloud Infrastructure dashboard with the search bar set to 'Japan East (Tokyo)'. On the left, there's a sidebar with 'リソース' (Resources) and two sections: 'イングレス・ルール(3)' and 'エクスレス・ルール(1)'. The main area is titled 'Ingress Rules' and shows a table with two existing rules. The first rule has a source CIDR of '0.0.0.0/0', a protocol of 'TCP', and a port range of 'All' to '22'. The second rule has a source CIDR of '0.0.0.0/0', a protocol of 'ICMP', and a port range of '3,4'. Below the table, there's explanatory text about SSH port forwarding. At the bottom of the table, there's a note: 'Copyright © 2020, Oracle and/or its affiliates. All rights reserved.'

▲図 3.32 [イングレス・ルールの追加] をクリック

[ソース CIDR] に [0.0.0.0/0]^{*10}を入力します。[宛先ポート範囲] には [80,443]^{*11}を入力します。(図 3.33) どちらも入力できたら、[イングレス・ルールの追加] をクリックします。

This screenshot shows the 'Add Ingress Rule' dialog box. It includes fields for 'Source CIDR' (set to '0.0.0.0/0') and 'Destination Port Range' (set to '80,443'). There are also dropdowns for 'Protocol' (set to 'TCP') and 'Source IP Type' (set to 'Any'). Below these fields is a 'Description' section with explanatory text about port forwarding. At the bottom of the dialog are 'Next Step' and 'Cancel' buttons.

▲図 3.33 [ソース CIDR] に [0.0.0.0/0]、[宛先ポート範囲] には [80,443] を入力

[イングレス・ルール] に、HTTP (80 番ポート) と HTTPS (443 番ポート) へのリクエストを通す設定が追加されました。(図 3.34)

*¹⁰ ソース CIDR は接続元の IP アドレス範囲のことで、0.0.0.0/0 はすべての IP アドレスを指します。つまり接続元がどんな IP アドレスでもファイアウォールを通れます、ということですね

*¹¹ ポート番号とは、サーバという家や、その手前のファイアウォールという壁についているドアのようなものだと思ってください。同じサーバを訪問するときでも SSH は 22 番のドアを、HTTP は 80 番のドアを、HTTPS は 443 番のドアを通ります

第3章 ウェブサーバの設定をしよう

The screenshot shows the OCI Network Services - Ingress Rules interface. A new rule has been added, highlighted with a red box. The rule details are as follows:

ステートレス	ソース	IPプロトコル	ソース・ポート範囲	宛先ポート範囲	タイプとコード	許可	説明
いいえ	0.0.0.0	TCP	All	22	ポートのTCPトラフィック 22 SSH Remote Login Port tcpall		
いいえ	0.0.0.0	ICMP			3, 4		次に対するICMPトラフィック ソース3, 4番ポートに到達でき ません。4番ポートへの入 り口が必要です。フラン クションが必要ですが、フラ グメントしないように設定さ れていました。
いいえ	10.0.0.0/16	ICMP			3		次に対するICMPトラフィック ソース3番ポートに到達でき ません。
いいえ	0.0.0.0	TCP	All	80	ポートのTCPトラフィック 80		
いいえ	0.0.0.0	TCP	All	443	ポートのTCPトラフィック 443 HTTPS		

▲図 3.34 ルールが追加された！

3.7.2 今度こそ HTTP でサイトを見てみよう

再び、サーバを立てたときにメモした [パブリック IP アドレス] を、ブラウザで開いてみましょう。(図 3.35)



Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

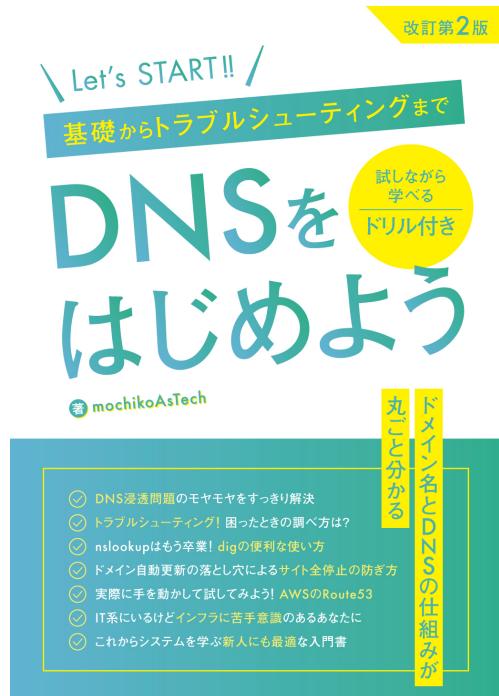
Thank you for using nginx.

▲図 3.35 HTTP でサイトが見られた！

[Welcome to nginx!] と表示されました！ これでまず、「HTTP でサイトを表示する」はクリアです。

3.8 ドメイン名の設定をしよう

ウェブサーバの準備ができたので、HTTPS のサイト用にドメイン名を用意します。「自分のドメイン名？ そんなの持っていないよ！」という人は、先に「DNS をはじめよう」(図 3.36) で、ドメイン名を買ってからこの先へ進むようにしてください。



▲図 3.36 「DNS をはじめよう」(1,000 円) は BOOTH や Amazon (Kindle) で好評発売中

あなたが買ったドメイン名を使って「ssl. 自分のドメイン名」の A レコードを作成して、サーバの [パブリック IP アドレス] と紐付けてください。ネームサーバはお名前.com を使用してもいいですし、AWS の Route53 で設定しても構いません。

なお筆者が「DNS をはじめよう」で購入したのは startdns.fun というドメイン名だったので、ssl.startdns.fun という A レコードを作って、さっ立てたばかりのサーバの [パブリック IP アドレス] と紐付けます。例えばネームサーバが「お名前.com」なら、DNS 設定の画面でこのように A レコードを追加します。(図 3.37)

第3章 ウェブサーバの設定をしよう

ホスト名	TYPE	TTL	VALUE				優先	状態	追加
ssl .startdns.fun	A	3600	140	238	.33	.51		有効	追加

▲図 3.37 「ssl. 自分のドメイン名」の A レコードを作成する

A レコードを追加できたかどうかは、次の dig コマンドで確認できます。dig コマンドをたたいた結果、サーバの IP アドレスが返ってくれば A レコードは設定できています。

```
$ dig ssl. 自分のドメイン名 a +short
```

筆者の場合は、次のように表示されました。

```
$ dig ssl.startdns.fun a +short  
140.238.33.51
```

ドメイン名が設定できたら、ブラウザでも「http://ssl. 自分のドメイン名」を叩いてみましょう。先ほどと同じ NGINX のページが表示されるはずです。（図 3.38）



Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

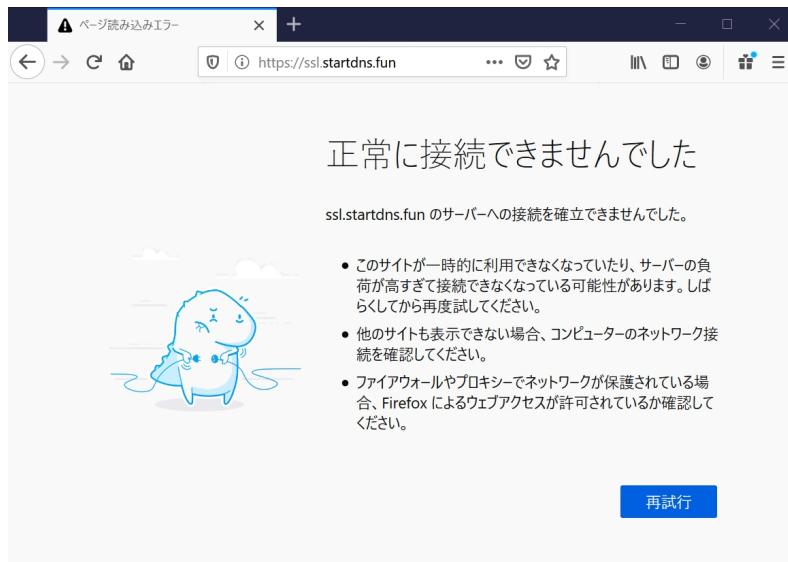
For online documentation and support please refer to nginx.org. Commercial support is available at nginx.com.

Thank you for using nginx.

▲図 3.38 「ssl. 自分のドメイン名」でサイトが表示された！

HTTP でサイトを見ることができましたが、同じドメイン名を HTTPS で開いてみるとどうでしょう？ ブラウザで「https://ssl. 自分のドメイン名」を開いてみると、「正常に接続できませんでした」と表示（図 3.39）されました。

3.8 ドメイン名の設定をしよう



▲図 3.39 HTTPS で開くと [正常に接続できませんでした] と表示された

それでは HTTPS でもサイトが見られるように、SSL 証明書を取得して設定をしていきましょう。

第4章

SSL 証明書を取得しよう

HTTP でサイトが見られたので、今度は HTTPS でも見られるよう、必要な材料を「SSL 証明書」を入手しましょう。

4.1 SSL 証明書にまつわる登場人物

SSL 証明書は、関わってくる登場人物が多いので、最初に登場人物一覧をご紹介します。

- 秘密鍵 (startssl.key)
- CSR (startssl.csr)
- SSL 証明書 (server.crt)
- 中間 CA 証明書 (ca-bundle.ca)
 - SSL 証明書+中間 CA 証明書 (startssl.crt)

これらの登場人物が集まる場所として、サーバの中で /etc/nginx/ の下に ssl というディレクトリを作っておきます。^{*1}

```
# mkdir /etc/nginx/ssl/  
# cd /etc/nginx/ssl/
```

4.2 秘密鍵 (startssl.key) を作ろう

先ず最初に作成するのが秘密鍵です。秘密鍵は「.key」や「.pem」^{*2}という拡張子で作成されることが多いです。秘密鍵は名前のとおり「秘密」にすべきです。つまり、限られた人だけが触れるように管理すべきです。決してメールに添付して送ったり、サーバ内で誰でも見られる/tmp/以下に置いたりしてはいけません。

次の openssl コマンドを叩いて、秘密鍵を生成しましょう。左から順に「openssl コマンドで、鍵アルゴリズムが RSA で、鍵の長さは 2048 ビットで、/etc/nginx/ssl/以下に startssl.key というファイル名で秘密鍵を作って」という意味です。

```
# openssl genrsa 2048 >/etc/nginx/ssl/startssl.key  
Generating RSA private key, 2048 bit long modulus  
.....+++
```

^{*1} もし `mkdir: cannot create directory '/etc/nginx/ssl': Permission denied` と表示されてしまったら、あなたはいま、うっかり一般ユーザのまま mkdir コマンドを実行しています。コマンドの例で、左側のプロンプトが「#」のときは、root で実行してください。「sudo su -」と書いて Enter キーを押すと root になれます

^{*2} 秘密鍵が PEM と呼ばれるテキスト形式で生成されることから、pem という拡張子が使われるようす

```
e is 65537 (0x10001)
```

できあがった秘密鍵を、cat コマンドで見てみましょう。

```
# cat /etc/nginx/ssl/startssl.key
-----BEGIN RSA PRIVATE KEY-----
MIIEpQIBAAKCAQEAO+s/Gzdb0bzg6QWzCvK5JofMv6izHzlCfMCMhcU7SeBd2tHN
icRA7g5CZq09aaEqv1949cFX5C3bgHx140+epeudrKyUjRwZSpS70mznDBFQByTY
(中略)
InsCw9qu+iZknMKiISw3Krht/898/hq0jqLFJUTbfg9BP8w+JWW4+8hp40Sklymc
NRcvPYUBQy3wK+w527rksodkGZ77c6Q+XxRtH/wpo3H+xwhmJvi+T2o=
-----END RSA PRIVATE KEY-----
```

【コラム】SSL 証明書の秘密鍵にパスフレーズは設定すべき？

openssl コマンドで秘密鍵を作るとき、-aes128 や-aes256 というオプションを付けると、パスフレーズを聞かれて、指定の暗号で暗号化された秘密鍵が output されます。^{*3} [Enter pass phrase:] や [Verifying - Enter pass phrase:] と表示された際に、いくらキーボードを叩いて入力しても、黒い画面上は何も表示されません。ですが、ちゃんと文字入力はできているので大丈夫です。入力を間違えたら Backspace キーで消して、書き直すこともできます。

```
# openssl genrsa -aes128 2048 >/etc/nginx/ssl/with-passphrase.key
Generating RSA private key, 2048 bit long modulus
(中略)
Enter pass phrase: ←秘密鍵のパスフレーズとして「startssl」を入力
Verifying - Enter pass phrase: ←もう一度「startssl」を入力
```

秘密鍵がパスフレーズで保護されていると、秘密鍵を盗んだ誰かが使おうとしても、パスフレーズが分からなければ使えないで安心です。

しかし秘密鍵にパスフレーズが設定されていると、Apache や Nginx といったウェブサーバを再起動した際にも、必ずパスフレーズを聞かれます。もし何かトラブルがあってサーバが OS ごと再起動してしまった場合、折角 Nginx が自動起動する設定になっていても、パスフレーズを聞くところで止まって起動できず、サイトが自動復旧しない、というトラブルが発生します。

これを回避するには、パスフレーズをファイルに書いておいて、Nginx の自動起動時にそれを読み込むようにする、という方法があります。しかし折角設定し

たパスフレーズをファイルに書いて同じウェブサーバ内に置いてしまうと、秘密鍵を盗むとき一緒にパスフレーズのファイルも盗まれてしまう可能性が高くなり、もはやパスフレーズを設定した意味がありません。そのため筆者は、ウェブサーバに設置して使う秘密鍵にはパスフレーズは設定しなくてよい、という考えです。

ちなみに、次のように一度パスフレーズありで作った秘密鍵を、パスフレーズなしで複製することも可能です。

```
# cd /etc/nginx/ssl/
# openssl rsa -in with-passphrase.key -out without-passphrase.key
Enter pass phrase for with-passphrase.key: ← 秘密鍵のパスフレーズ
「startssl」を入力
writing RSA key
```

利便性を保つつつ安全性も向上させたければ、本番のウェブサーバで使う秘密鍵はパスフレーズなし、バックアップサーバで保管しておく秘密鍵はパスフレーズありにしておく、という方法がいいでしょう。

4.3 CSR (startssl.csr) を作ろう

秘密鍵ができたら、続いて CSR (Certificate Signing Request) の作成に進みましょう。CSR は「.csr」という拡張子で作成されることが多いです。CSR は「Certificate Signing Request (証明書署名リクエスト)」という名前のとおり、認証局に対して「こういう内容で SSL 証明書を作って署名して」とリクエストする、申請書のようなものです。

左から順に、「openssl コマンドで、CSR を、新しく、秘密鍵は startssl.key で、startssl.csr というファイル名で作って」という意味です。

```
# cd /etc/nginx/ssl/
# openssl req -new -key startssl.key -out startssl.csr
```

CSR を作成するため、いくつか質問をされます。(表 4.1)^{*4}

^{*3} さっきは-aes128 や-aes256 といった「どの暗号で暗号化するか?」のオプションを何も指定しなかったので、パスフレーズは聞かれず、秘密鍵も暗号化されませんでした

^{*4} 本著では DV 証明書を取得するため「組織単位名 (OU)」は任意としていますが、証明書の種類や認証局によっては必須のところもあるようです。取得時に認証局のサイトで確認しましょう。DV 証明書については後述します

4.3 CSR (`startssl.csr`) を作ろう

▼表 4.1 CSR 作成時に聞かれる質問

必須/任意	質問	説明	入力する内容
必須	Country Name	国名 (C)	JP
必須	State or Province Name	都道府県名 (S/ST)	Tokyo
必須	Locality Name	市町村名 (L)	Shinjuku-ku
必須	Organization Name	組織名 (O)	mochikoAsTech
任意	Organizational Unit Name	組織単位名 (OU)	入力なし
必須	Common Name	コモンネーム (CN)	ssl.startdns.fun

```
Country Name (2 letter code) [XX]:JP
State or Province Name (full name) []:Tokyo
Locality Name (eg, city) [Default City]:Shinjuku-ku
Organization Name (eg, company) [Default Company Ltd]:mochikoAsTech
Organizational Unit Name (eg, section) []:
Common Name (eg, your name or your server's hostname) []:ssl.startdns.fun
```

次の 3 つは入力不要です。

```
Email Address []:
A challenge password []:
An optional company name []:
```

質問に全て答えると、CSR ができあがります。「Subject」と書かれた行を見て、生成された CSR の内容が正しいか確認しましょう。

```
# openssl req -text -in /etc/nginx/ssl/startssl.csr -noout | head -4
Certificate Request:
Data:
Version: 0 (0x0)
Subject: C=JP, ST=Tokyo, L=Shinjuku-ku, O=mochikoAsTech, CN=ssl.startdns.fun
```

CSR を cat コマンドで表示して、「-----BEGIN CERTIFICATE REQUEST-----」から「-----END CERTIFICATE REQUEST-----」までをメモしておきましょう。メモした CSR はこの後使用します。

```
# cat /etc/nginx/ssl/startssl.csr
-----BEGIN CERTIFICATE REQUEST-----
MIICqzCCAZMCAQAwZjELMAkGA1UEBhMCS1AxDjAMBgNVBAgMBVRva3lvMRQwEgYD
```

```
VQQHDAtTaGluanVrdS1rdTEWMBQGA1UECgwNbW9jaGlrbOFzVGVjaDEZMBcGA1UE  
(中略)  
jP1RMQS3PuYDE6QIVJ5zbMC+RIydSQ/ODr9VUHWiYqDPjx+BpphYT5AxMwbw9/m5  
noKGVJ11Mt7G03Awa/TX  
-----END CERTIFICATE REQUEST-----
```

4.3.1 【ドリル】CSRで入力すべきなのはクライアントの情報？

問題

A銀行のウェブサイトをHTTPSで作ることになりました。SSL証明書^{*5}の取得はA銀行の代わりに広告代理店のB社が行い、さらにサイトの制作や運用はA銀行からWeb制作会社のC社に委託する場合、CSRで入力する住所や会社名はA銀行・B社・C社のどれにすべきでしょうか？

- A. A銀行のウェブサイトなんだからA銀行を入力すべき
- B. A銀行から任されてSSL証明書を買うのはB社だからB社を入力すべき
- C. 実際にサイトの管理を任せているのはC社だからC社を入力すべき

答え _____

解答

正解はAです。ウェブサイトの運営元がA銀行であることを証明するためのSSL証明書なので、CSRではA銀行（クライアント）の情報を記載すべきです。

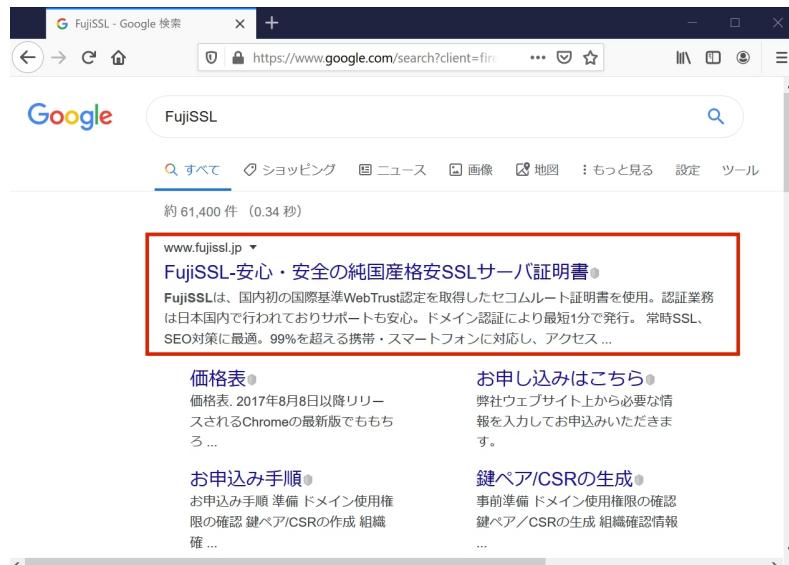
A銀行のフィッシングサイトが出てきたときに、エンドユーザが「本物のサイトか確認しよう」と思って証明書の情報を見て、B社やC社の情報が表示されたら「A銀行じゃない！」となってしまいます。

4.4 SSL 証明書の取得申請を出そう

[FujiSSL]で検索して、[FujiSSL-安心・安全の純国産格安SSLサーバ証明書]をクリック（図4.1）します。

^{*5} SSL証明書の種類は「EV証明書」とします。EV証明書については後述します

4.4 SSL 証明書の取得申請を出そう



▲図 4.1 [Fujissl-安心・安全の純国産格安SSLサーバ証明書] をクリック

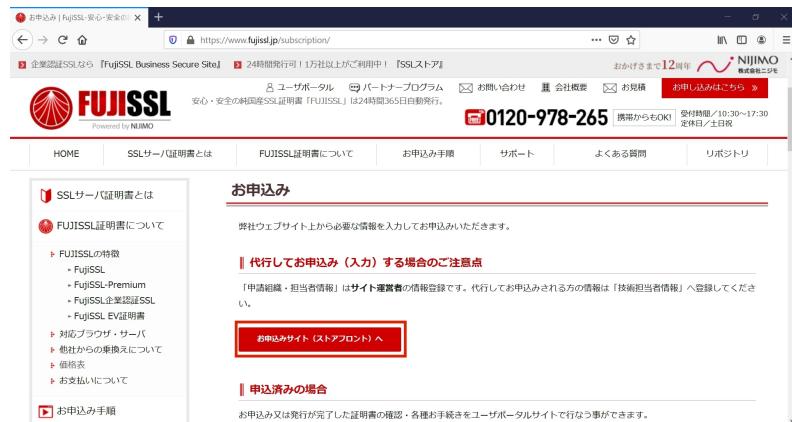
右上の「お申し込みはこちら」をクリック（図 4.2）します。



▲図 4.2 [お申し込みはこちら] をクリック

続いて「お申し込みサイト（ストアフロント）へ」をクリック（図 4.3）します。

第4章 SSL証明書を取得しよう



▲図4.3 [お申し込みサイト（ストアフロント）へ] をクリック

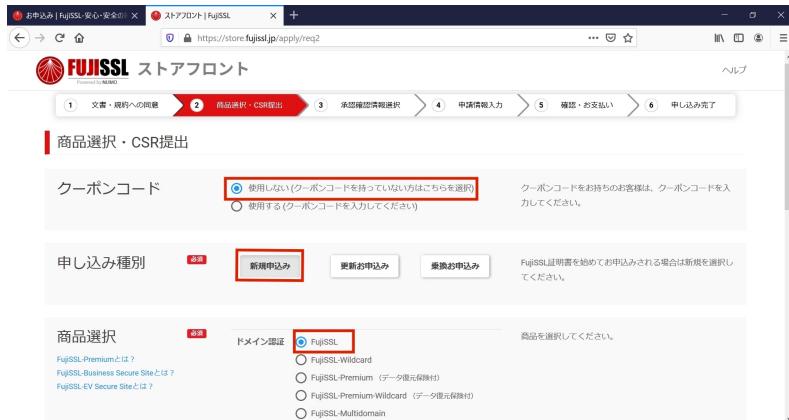
指定された文書と「収集した個人情報の利用目的」を確認した上で、チェックボックスにチェックを入れて [次へ] をクリック（図4.4）します。



▲図4.4 チェックを入れて [次へ] をクリック

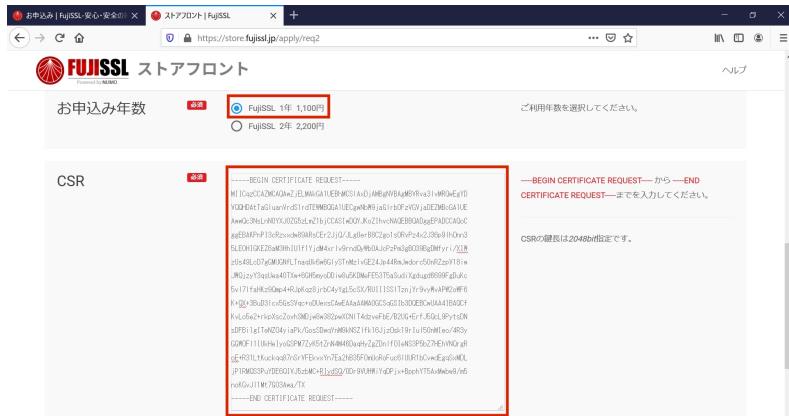
クーポンコードは [使用しない]、申し込み種別は [新規申し込み]、商品選択は [ドメイン認証] の [FujisSL] になっていることを確認（図4.5）します。

4.4 SSL 証明書の取得申請を出そう



▲図 4.5 クーポンコード、申し込み種別、商品選択を確認

下へスクロールして、お申込み年数が「[FujiSSL 1年 1,100円]」^{*6}になっていることを確認（図 4.6）したら、CSR に、先ほど「-----BEGIN CERTIFICATE REQUEST-----」から「-----END CERTIFICATE REQUEST-----」までをメモしておいた CSR をペーストします。

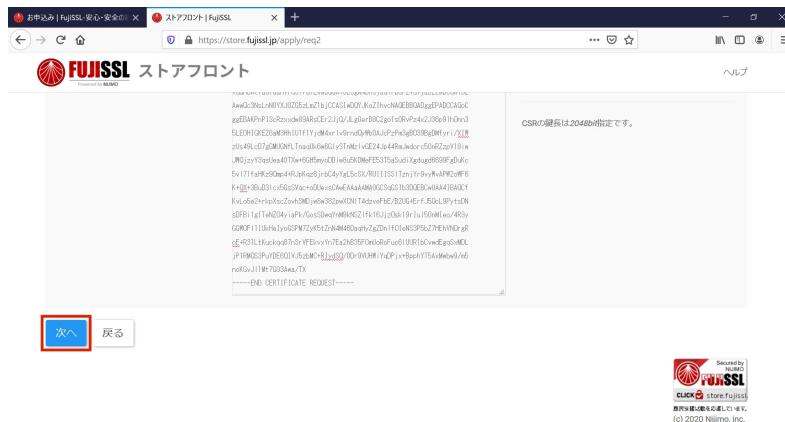


▲図 4.6 CSR をペーストする

[次へ] をクリック（図 4.7）してください。

^{*6} 2019年2月現在、Fujissl の「ドメイン認証シングルタイプ」というSSL証明書は、有効期間1年で税込1,100円です

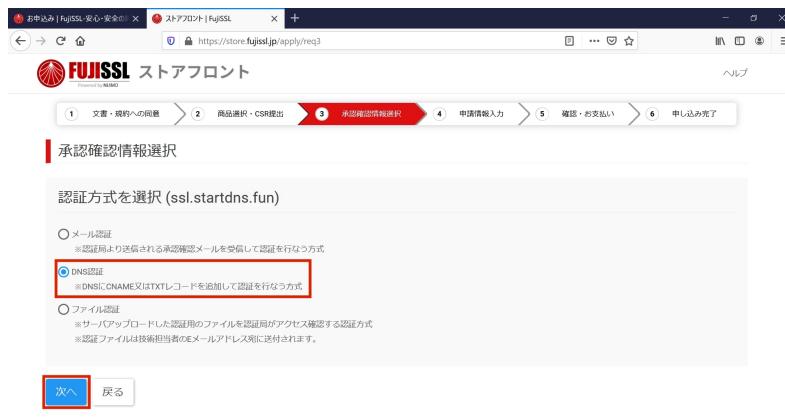
第4章 SSL証明書を取得しよう



▲図4.7 [次へ] をクリック

認証方式は「DNS認証」を選択（図4.8）してください。これは「CSRのコモンネームで指定したドメイン名が、あなたの持ち物であることをどうやって証明しますか？」ということを聞かれています。対象のドメイン名で、メールを受信してURLを踏むか、指定された内容のリソースレコードをDNSで追加するか、サイトに指定された内容のファイルをアップするか、いずれかの方法で「このドメイン名（筆者ならssl.startdns.fun）は私の持ち物です」ということを証明しなければいけません。

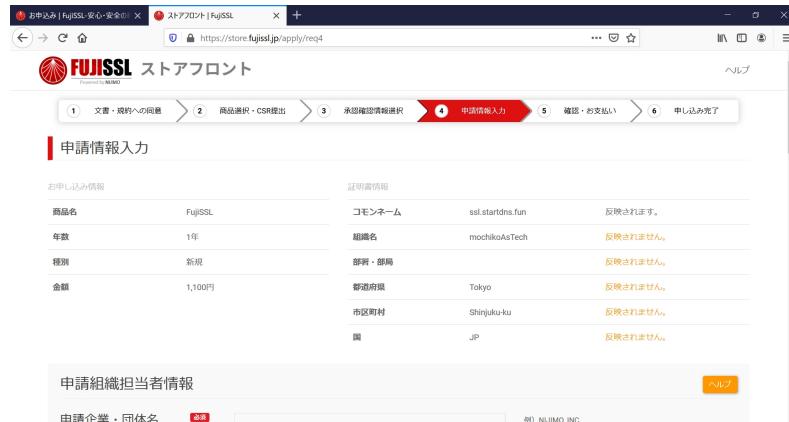
今回はDNSでTXTレコードを追加する、という方法で証明するので、「DNS認証」を選択して「次へ」をクリックします。



▲図4.8 [DNS認証] を選択して [次へ] をクリック

4.4 SSL 証明書の取得申請を出そう

今回購入するのは DV 証明書です。DV 証明書は「そのドメイン名の使用権があること」だけを証明してくれます。サイトの運営者が日本にいることや、東京の新宿区にオフィスがあること、mochikoAsTech という組織であることなどは確認も証明もしないので、実際の証明書にもそれらの情報は反映されません。（図 4.9）



▲図 4.9 [申請情報入力] 画面

[申請組織担当者情報] と [技術担当者情報] を英語表記で入力（図 4.10）します。ここで入力した住所や電話番号、個人名などは外向けに公開されることはありませんので、安心して入力してください。^{*7}この後、ここで入力した [E メールアドレス] 宛てに、SSL 証明書が送られてきます。メールアドレスを間違えないよう注意してください。すべて入力したら、[次へ] をクリック（図 4.11）します。

^{*7} EV 証明書や OV 証明書の場合は、ここで入力した担当者宛てに実在確認の連絡がります。詳細については後述します

第4章 SSL 証明書を取得しよう

申請組織担当者情報

申請企業・団体名
※ 英語表記
例) Nujmo, INC.
※個人の場合は個人名を入力してください。

お名前
※ 英語表記
姓 Nekomura 名 Mochiko
例) 姓 : Fuji 名 : Taro

役職
※ 英語表記
例) CEO

国名
日本 (Japan)

郵便番号
例) 000-0000

所在地
※ 英語表記
例) Tokyo

詳細
Shibuya-ku
例) Shibuya-ku

4-1-6 Shinjuku
例) 1-12-2, Shibuya
例) CROSS OFFICE SHIBUYA 5F

▲図 4.10 [申請組織担当者情報] と [技術担当者情報] を入力

所在地
※ 英語表記
例) Tokyo

Shibuya-ku
例) Shibuya-ku

4-1-6 Shinjuku
例) 1-2-2, Shibuya

SHINJUKU MIRAINA TOWER
例) CROSS OFFICE SHIBUYA 5F

電話番号
例) 03-XXXX-XXXX

Eメールアドレス
例) taro@fujssl.jp
証明書送付先メールアドレスになります。

次へ 戻る

▲図 4.11 [次へ] をクリック

[決済情報入力] に、SSL 証明書代を支払うクレジットカード情報を入力（図 4.12）します。

4.4 SSL 証明書の取得申請を出そう

The screenshot shows a web browser window for 'FujSSL Storefront'. The main title is '決済情報入力' (Payment Information Input). It displays a payment amount of '1,100 円' with a note '(※税込み)'. Below this, there's a section for 'ご利用いただけるカードの種類' (Card Types Available) with icons for American Express, MasterCard, VISA, JCB, Diners Club, and UnionPay. The 'クレジットカード番号' (Credit Card Number) field is filled with a redacted number and includes a note about expiration dates. The '有効期限' (Expiration Date) field shows a dropdown menu for month and year. The 'セキュリティコード' (Security Code) field is also redacted. The 'カード名義' (Cardholder Name) field contains 'TARO FUJI'. At the bottom, there's a section for '書類送付先' (Document Delivery Address) which is currently empty.

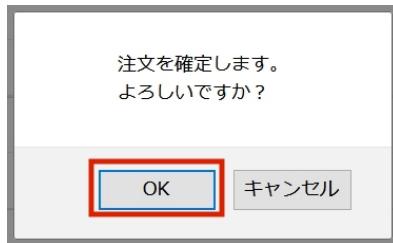
▲図 4.12 クレジットカード情報を入力

【書類送付先】で【請求書宛名】、【納品書宛名】、【領収書宛名】を記入したら、【上記の内容で注文を確定する】をクリック（図 4.13）します。

This screenshot shows the 'Document Delivery Address' section of the FujSSL storefront. It lists three fields: '請求書宛名' (Bill of Lading Recipient), '納品書宛名' (Delivery Note Recipient), and '領収書宛名' (Receipt Recipient), each containing the value 'mochikoAsTech'. Below these fields is a button labeled '上記の内容で注文を確定する' (Confirm Order with Above Content). At the bottom right, there's a security seal from 'CLICK & STORE FujSSL'.

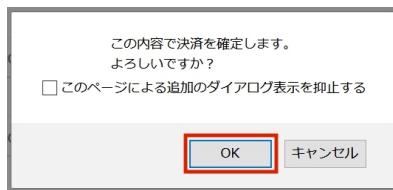
▲図 4.13 【上記の内容で注文を確定する】をクリック

「注文を確定します。よろしいですか？」と表示（図 4.14）されたら、[OK] をクリックします。



▲図 4.14 [OK] をクリック

「この内容で決済を確定します。よろしいですか？」と表示（図 4.15）されます。「1年間有効な SSL 証明書を 1,100 円で買うんだ！」というケツイ^{*8}をしたら、[OK] をクリックします。



▲図 4.15 1,100 円払うケツイをして [OK] をクリック

お申し込み完了のページが表示（図 4.16）されました。先ほど登録したメールアドレス宛に、DNS 設定情報を知らせるメールが届いていますので確認しましょう。

^{*8} UNDERTALE というゲームは、ゲームオーバーになるたび「ケツイを ちからに かえるんだ…！」というメッセージが表示されます。筆者はパビルスが好きですが、初見でうっかり殺してしまい、同僚に人でなし扱いされました

4.5 DNS の設定をしよう

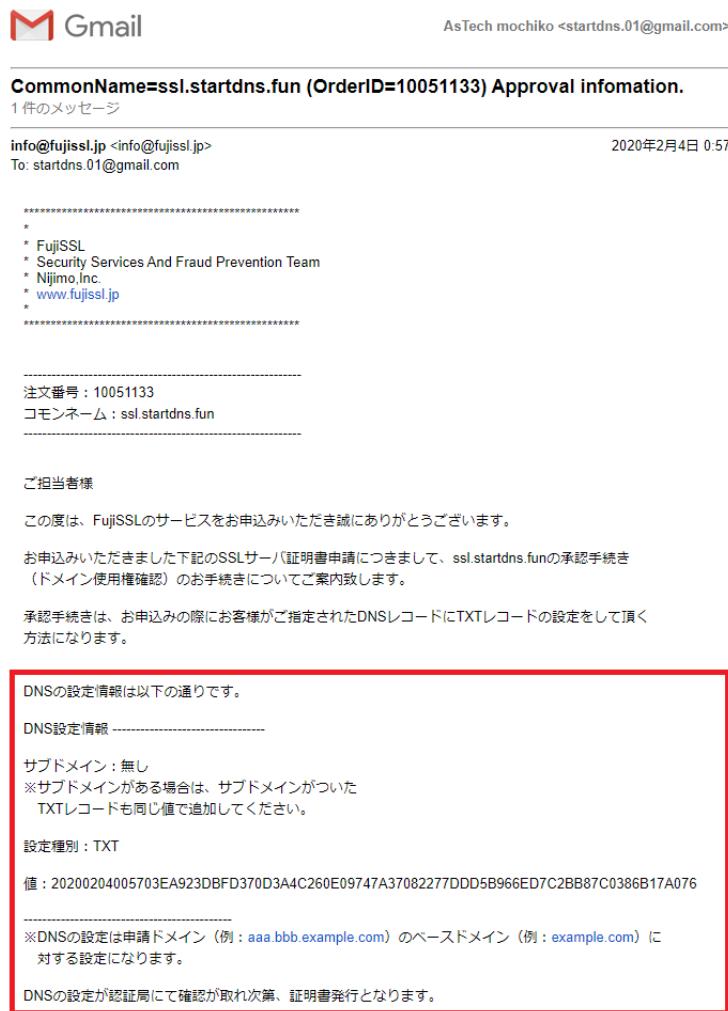


▲図 4.16 [お申込み完了] 画面

4.5 DNS の設定をしよう

[CommonName=ssl. 自分のドメイン名] から始まる件名のメール（図 4.17）がすぐに届きました。「CSR のコモンネームで指定したドメイン名が ssl.startdns.fun の場合、startdns.fun の TXT レコードを追加して、メールに書いてある値を設定するよう書いてあります。

第4章 SSL証明書を取得しよう



▲図 4.17 追加すべき TXT レコードの値がメールで届いた

例えばネームサーバが「お名前.com」なら、DNS 設定の画面でこのように TXT レコードを追加（図 4.18）します。今回はサブドメインを含まないドメイン名を追加するので、[ホスト名] には何も入力しません。[VALUE] には、メールに書いてあった値をそのままコピペーストします。

4.5 DNS の設定をしよう

A/AAAA/CNAME/MX/NS/TXT/SRV/DS/CAAレコード						
ホスト名	TYPE	TTL	VALUE	優先	状態	追加
.startdns.fun	TXT ▼	3600	20200204005703EA923DBF		有効 ▼	追加

▲図 4.18 「自分のドメイン名」の TXT レコードを追加する

TXT レコードができたかどうかは、次の dig コマンドで確認できます。dig コマンドをたたいた結果、サーバの IP アドレスが返ってくれば A レコードは設定できています。

```
$ dig 自分のドメイン名 txt +short
```

筆者の場合は、次のように表示されました。

```
$ dig startdns.fun txt +short  
"20200204005703EA923DBFD370D3A4C260E09747A37082277DDD5B966ED7C2BB87C0386B17A076"
```

TXT レコードを追加してから、おおよそ 30 分後に SSL 証明書がメール（図 4.19）で届きました。

Gmail AsTech mochiko <startdns.01@gmail.com>

(OrderID=10051133) FujiSSL Fulfilment E-mail

2件のメッセージ

info@fujissl.jp <info@fujissl.jp> 2020年2月4日 1:30
To: startdns.01@gmail.com

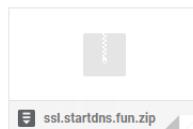
*
* FujiSSL
* Security Services And Fraud Prevention Team
* Nijimo,Inc.
* www.fujissl.jp
*

注文番号 : 10051133
コモンネーム : ssl.startdns.fun

ご担当者様
このたびはFujiSSLのサービスをご利用いただき誠にありがとうございます。
お客様の証明書が発行いたしましたのでご案内いたします。
証明書はこのメールの下部に貼付されております。
当メールの添付ファイルから証明書一式を取得いただくことも可能です。

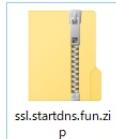
▲図 4.19 SSL 証明書がメールで届いた

メールに添付されている ZIP ファイル (ssl_自分のドメイン名.zip) に SSL 証明書が入っていますのでダウンロードします。(図 4.20)



▲図 4.20 メールに添付されている ZIP ファイル

Windows の方も Mac の方も、ダウンロードした ZIP ファイルはデスクトップに置いてください。(図 4.21)



▲図 4.21 ダウンロードした ZIP ファイルはデスクトップに置く

ZIP ファイルを右クリックして、[すべて展開] をクリックします。(図 4.22)



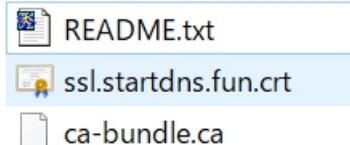
▲図 4.22 右クリックして [すべて展開] をクリック

[展開] をクリックします。(図 4.23)



▲図 4.23 [展開] をクリック

展開したフォルダ（図 4.24）の中の [server.crt] が SSL 証明書で、[ca-bundle.ca] が中間 CA 証明書です。README はファイルの説明書です。



▲図 4.24 server.crt が SSL 証明書で、ca-bundle.ca が中間 CA 証明書

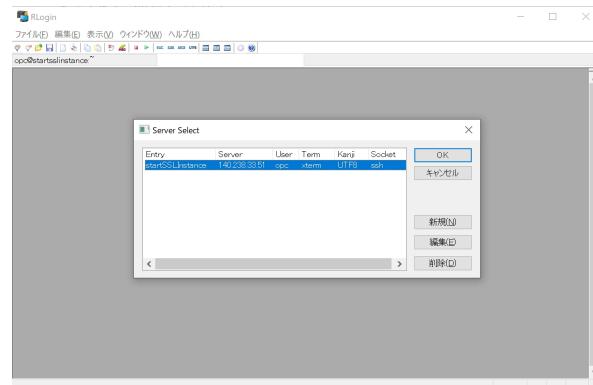
どちらも必要なものなので、この 2 つのファイルをサーバにアップロードしましょう。

4.6 SSL 証明書をサーバに設置しよう

それでは SSL 証明書をサーバに設置します。NGINX では SSL 証明書と中間 CA 証明書の 2 つを、1 ファイルにまとめて使用するので、まずはアップロードして、それから 1 ファイルにまとめる作業を行ないます。

4.6.1 Windows で証明書と中間 CA 証明書をアップしよう

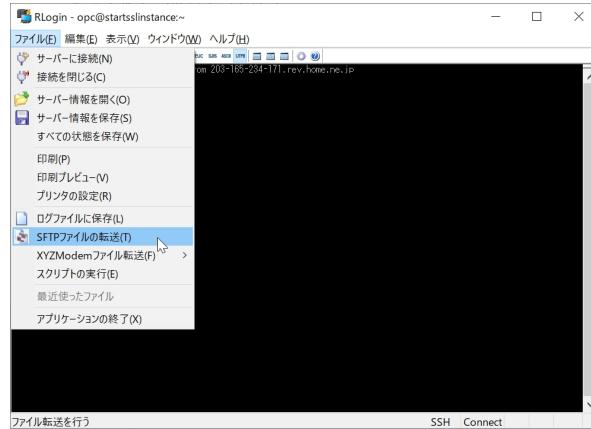
Windows のパソコンを使っている方は、RLogin を起動します。[startSSLInstance] を選択して、[OK] をクリック（図 4.25）します。



▲図 4.25 RLogin を起動して [startSSLInstance] を選択してログイン

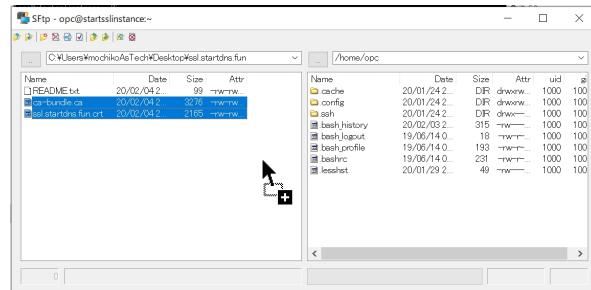
黒い画面が開いたら、[ファイル] から [SFTP ファイルの転送] をクリック（図 4.26）します。

4.6 SSL 証明書をサーバに設置しよう



▲図 4.26 [ファイル] から [SFTP ファイルの転送] をクリック

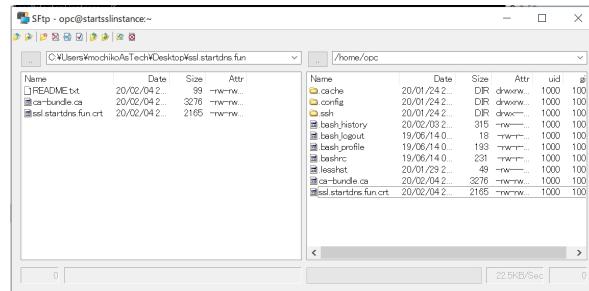
左側があなたのパソコンで、右側がサーバです。左側をデスクトップに展開したフォルダ^{*9}にしたら、[server.crt] と [ca-bundle.ca] を右側にドラッグ & ドロップ（図 4.27）してください。これでサーバの「/home/opc」にファイルがアップロードされます。



▲図 4.27 [ファイル] から [SFTP ファイルの転送] をクリック

右側に [server.crt] と [ca-bundle.ca] がアップされたら、×を押してファイル転送の画面は閉じて構いません。（図 4.28）

^{*9} 筆者の場合は「C:\Users\mochikoAsTech\Desktop\ssl.startdns.fun」でした



▲図 4.28 [ファイル] から [SFTP ファイルの転送] をクリック

4.6.2 Mac で証明書と中間 CA 証明書をアップしよう

Mac を使っている方は、ターミナルを起動してください。scp コマンドを使って、Mac の中にある [server.crt] と [ca-bundle.ca] を、サーバにアップロードします。[展開した フォルダ] と [パブリック IP アドレス] の部分は、ご自身のものに書き換えてください。

```
$ cd ~/Desktop/展開したフォルダ
$ scp -i ~/Desktop/startSSLKey server.crt ca-bundle.ca opc@パブリック IP アドレス:/home/opc/
server.crt    100%   0    0.0KB/s   00:00
ca-bundle.ca  100%   0    0.0KB/s   00:00
```

4.6.3 証明書を 1 ファイルにまとめよう

サーバにログインしている状態で、次のコマンドを叩いて、先ほどアップロードした [server.crt] と [ca-bundle.ca] が、ちゃんと「/home/opc/」以下に存在していることを確認します。

```
$ sudo su -
# ls /home/opc/
ca-bundle.ca  ssl.startdns.fun.crt
```

続いて 2 つのファイルから、新たに [startssl.crt] というファイルを作りましょう。^{*10}NGINX では SSL 証明書と中間 CA 証明書という 2 つのファイルを、1 つの

^{*10} このとき cat コマンドで結合すると、SSL 証明書と中間 CA 証明書の間に改行が入らず、「-----END

4.7 NGINX で HTTPS のバーチャルホストを作ろう

ファイルにがっちゃんこ！ とつなげて使うためです。

```
# cd /etc/nginx/ssl/
# awk 1 /home/opc/ssl.startdns.fun.crt /home/opc/ca-bundle.ca > startssl.crt
```

cat コマンドで [startssl.crt] を確認してみましょう。次のように「----BEGIN CERTIFICATE----」と「----END CERTIFICATE----」が、繰り返し 3 つ表示されれば大丈夫です。

```
# cat /etc/nginx/ssl/startssl.crt
-----BEGIN CERTIFICATE-----
MIIF/DCCBOoSgAwIBAgIQaoS/P1b4mCR8mn5/WUrI6zANBgkqhkiG9wOBAQsFADBn
MQswCQYDVQQGEwJKUDE1MCMGA1UEChMcUOVDT0OgVHJ1c3QgU3lzdGVtctyBDTy4s
(中略)
31pTIUPabkFxDPjs1Vw9c7z3Vgk2fnpuwE21rE+46zrJ3oTRqsABDbYreK1a5vsG
tcnpU1L1hrk/rC3JuI2ttHVaHU+1JCgTSRpv03a44azDy15T5C97dGxuowPgcaMQ
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
MIIEpzCCA4+gAwIBAgIJIrnxVPM8X14AMAOGCSqGSIB3DQEBCwUAMFOxCzAJBgNV
BAYTAkpQMSUwIwYDVQQKExxTRUNPTSBUcnVzdCBTeXNOZW1zIENPLixMVEQuMScw
(中略)
g3tiJA1fIpfxXD4cArZo6Z1XJ26B4H7vk5GmyR6poDy/CRvC7VIz3xp6o2348W1j
32S9pEuZhtxtMvPjnsHIWPNdz8pHv21x7bYwDnocwN2uk3Qrr1jxTQ9evg==
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
MIECjCCA1qgAwIBAgIJERmw+nLg2EjGMAOGCSqGSIB3DQEBCwUAMFAxCzAJBgNV
BAYTAkpQMRgwFgYDVQQKEw9TRUNPTSBUcnVzdC5uZXQxJzA1BgNVBAsTH1N1Y3Vy
(中略)
u5ZuCjxerxj3qS1rM46bcEfjopnaD7hnJXSYiL1d0yw5zSW2PEe+LHdoIAb2I6D8
8UFJH0C1i6sY518jhjk0Os1yeu1C/RcYO+NBNHKZkFEeEb6ez0sg=
-----END CERTIFICATE-----
```

これで証明書ファイルの準備は完了です。

4.7 NGINX で HTTPS のバーチャルホストを作ろう

「/etc/nginx/conf.d/」というディレクトリの下で、もともとあった設定ファイルをバックアップしておきます。

```
# cd /etc/nginx/conf.d/
# mv default.conf default.conf.backup
```

CERTIFICATE-----BEGIN CERTIFICATE----」のようになってしまふため、awk コマンドを使っています

第4章 SSL証明書を取得しよう

vi コマンドで、同じ場所に新しい設定ファイルを作ります。vi コマンドでファイルを編集するときは、i（アイ）を押してから入力します。書き終わったら ESC キーを押して、「:wq」と入力して Enter キーを押せば変更が保存されます。

```
# vi startssl.conf
server {
    listen 80 default_server;
    return 301 https://$host$request_uri;
}

server {
    listen      443 ssl http2;
    server_name ssl.startdns.fun;

    # 秘密鍵
    ssl_certificate_key /etc/nginx/ssl/startssl.key;
    # SSL証明書+中間CA証明書
    ssl_certificate      /etc/nginx/ssl/startssl.crt;

    # 暗号スイート
    ssl_ciphers ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-AE
    # プロトコルバージョン
    ssl_protocols       TLSv1.2;
    # 暗号スイートの順序はサーバが決める
    ssl_prefer_server_ciphers   on;

    location / {
        root   /usr/share/nginx/html;
        index  index.html index.htm;
    }
}
```

設定ファイルが書けたら、構文エラーがないかテストをします。もし書き間違いがあれば、ここでエラーメッセージとして表示されます。

```
# nginx -t
nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
nginx: configuration file /etc/nginx/nginx.conf test is successful
```

[test is successful] と表示されたら、NGINX を再起動して設定を反映しましょう。

```
# systemctl restart nginx.service
```

それからアクセスしたときに表示される、index.html の文字も [Let's start SSL/TLS] に変えておきましょう。

```
# cd /usr/share/nginx/html/
# cp -p index.html index.html.backup
# echo "Let's start SSL/TLS" > index.html
```

4.8 HTTPS でサイトを開いてみよう

証明書を設置して、NGINX の設定ファイルを修正し、NGINX の再起動もしたのでブラウザで「http://ssl.自分のドメイン名」を開いてみましょう。鍵マーク付きで HTTPS のページが表示されるはずです。（図 4.29）



▲図 4.29 HTTPS でサイトが表示された！

【コラム】ロードバランサーで SSL ターミネーションする方法もある

なお本著ではウェブサーバに SSL 証明書を設置しましたが、ウェブサーバの手前にロードバランサを置いて、そこに SSL 証明書を設置して、SSL ターミネーションを行う方法もあります。その場合、SSL で通信するのはエンドユーザのパソコンから終端となるロードバランサーまでで、ロードバランサーとウェブサーバの間は HTTP で通信するのが一般的です。

この方法には、次のようなメリットがあります。

- 暗号化や復号の処理を行う終端がロードバランサになるので、ウェブサーバの処理負荷が下がる
- (AWS の ELB と ACM の場合は) 証明書の取得や更新、設置が自動で行わ

れる

第5章

SSL/TLSについて学ぼう

SSL 証明書を取得して、実際に HTTPS のサイトを公開できました。この章では実践と照らし合わせながら SSL/TLS について学んでいきます。

5.1 SSL/TLSとは？

SSL (Secure Socket Layer) /TLS (Transport Layer Security) とは、インターネット上で安全にデータを送受信するための約束事（プロトコル¹）です。

SSLもTLSもあくまでプロトコル、つまり通信するための「約束事」なので、実際に通信するときは、そのプロトコルに従って実装されたソフトウェアを使います。SSL/TLSでは、OpenSSLというオープンソースのソフトウェアを使うことが殆ど²です。

5.1.1 SSLとTLSはどういう関係？

SSLとTLSは別々の名前ですが、その役割に大きな違いはありません。「SSL3.0」からバージョンアップする際に、「SSL3.1」ではなく「TLS1.0」という新しい名前が付けられました。つまり「TLSはSSLの後継バージョン」ということです。

ちなみに名前がまだSSLだったときの最後のバージョン「SSL3.0」は、重大な脆弱性³が見つかり、2015年のRFC7568⁴でもう使わないよう「Do Not Use SSL Version 3.0」と示されています。

ですがSSLという言葉の認知度が高く、TLSと呼んでもピンとこない人の方が多いため、現状は分かりやすさと正確さを両立させてSSL/TLSと併記することが多いです。

本著ではSSL/TLSのことを指して、SSLという言葉を使用しています。

5.1.2 SSLイコールHTTPSではない

サイト全体を「https://」から始まるURLにすることを、「常時SSL化」のように呼ぶため、皆さんの中にはSSL=HTTPSだと思っている人がいるかも知れません。

HTTPとSSLを組み合わせて使うことで、通信を保護するプロトコルがHTTPS(HTTP over SSL/TLS)です。ですが、SSLはHTTPSにおいてのみ使われるプロトコルではありません。例えばサーバにファイルをアップするときのFTPとSSLを組み合わせて使うFTPS(FTP over SSL)や、メール送信のSMTPとSSLを組み合わせて使

*¹ 例えば手紙は「メッセージを書いた便せんを封筒に入れて、表に宛先、裏に差出人を書き、重さに応じた切手を貼ってポストに入れる」という取り決めに従えば、きちんと相手に届きます。手紙の例と同じように、インターネットで通信を行う際、どんなデータをどんな方法で送受信するのか、という「約束事」のことをプロトコルと呼びます。SSLもTLSもプロトコルですし、HTTPやHTTPS、DNSもSMTPもプロトコルです

*² SSL証明書を取得するために、秘密鍵やCSRを作ったときのコマンドも、opensslコマンドでしたね

*³ 脆弱性（ぜいじゃくせい）というのは悪用が可能なバグや設定不備のことです

*⁴ Deprecating Secure Sockets Layer Version 3.0 <https://tools.ietf.org/html/rfc7568>

う SMTPS (SMTP over SSL) など、HTTPS 以外にも SSL が使われている場面はたくさんあります。

繰り返しになりますが、SSL はインターネット上で安全にデータを送受信するためのプロトコルです。上位のアプリケーション層^{*5}が HTTP であれ、FTP であれ、SSL を組み合わせて使うことでデータを安全に送受信できるようになります。

SSL = HTTPS ではない、ということを踏まえた上で、ここからは「サイトを HTTPS 化する」ことについて学んでいきましょう。

5.2 「サイトを HTTPS 化する」とは何か？

そもそもですが「サイトを HTTPS 化する」とは、なんでしょう？

本著では、「サイトの HTTPS 化」を、**ウェブサイト全体を「https://」から始まる URL にすること**と定義します。これは「常時 SSL 化」や「常時 SSL/TLS 化」、「フル SSL 化」、「AOSSL (Always On SSL の略)」といった言葉で表現されることもあります。

2014 年ごろはまだ、お問い合わせや会員登録など、個人情報を入力したり表示したりする一部のページのみを HTTPS にしているサイトが多く存在^{*6}していました。ですが Google が HTTPS を強く推奨はじめたことで、「一部ページだけに限らず、ウェブサイト全体を HTTPS にしよう」という動きが段々と活発になっていきました。

大手サイトが次々と HTTPS 化をはじめたのが、2016 年から 2017 年ごろだったと記憶しています。たとえばクックパッドは 2017 年 1 月^{*7}、日経新聞電子版は 2017 年 8 月^{*8}に、それぞれ HTTPS 化が完了したことを発表しています。

そして 2020 年現在、HTTPS 化の動きは、大手サイトだけでなくあらゆるサイトを対象にさらに進みつつあります。

5.3 どんなサイトでも必ず HTTPS にしなきゃだめ？

でも企業がやっているネットショップならまだしも、個人情報をやり取りするわけでもない、ただ個人の日記を公開しているだけのサイトも、HTTPS 化しなければいけないのでしょうか？

確かにサイトを HTTPS 化すると「通信が保護される」という大きなメリットがありま

^{*5} OSI 参照モデルのアプリケーション層のこと。エンジニア諸氏は誰しも、大学の授業や新卒研修で一度は「アセトネデブ」という語呂合わせを聞いたことがあるのでは…

^{*6} <https://techlife.cookpad.com/entry/2017/04/19/190901>

^{*7} <https://techlife.cookpad.com/entry/2017/04/19/190901>

^{*8} <https://www.nikkei.com/topic/20170808.html>

す。2014年より前は「HTTPS化すれば通信が保護される。けれど特に保護すべきやりとりでなければ、HTTPS化しなくても悪いことが起きる訳ではない」という状況でした。

5.4 HTTPのままだと起きるデメリット

ですが2020年現在は、HTTPS化しないでいるとさまざまなデメリットが発生します。どんなデメリットが起きるのか、具体的に解説していきましょう。

5.4.1 サイトが「安全でない」と表示されてしまう

GoogleはHTTPからHTTPSへの移行を強く推進しています。その施策の1つとして、Googleが提供するウェブブラウザ「Google Chrome」では、2018年7月にリリースされたバージョン68から、HTTPSでないページに対して「保護されていない通信」という表示をするようになりました。

Chromeでサイトを開いて、URLが「http://」で始まるものだった場合、次のようにURLの左側に「保護されていない通信」と表示（図5.1）されます。



▲図5.1 いきなり！ステーキのサイトを開くと「保護されていない通信」と表示される

サイトを見たとき、URLの真横に「保護されていない通信」と表示されたら、「なんだか危なさそう…見ない方がいいのかも…？」と心配になってしまいますよね。ChromeだけでなくFirefoxも、HTTPのサイトに対して鍔前に赤い斜め線が入ったマークの表示（図5.2）を行っており、サイトをHTTPS化しないことで、エンドユーザにサイトが「安全でない」と思われるてしまう状況にあります。



▲図5.2 HTTPのサイトを開くと「保護されていない通信」と表示される

5.4.2 Wi-Fi スポットでセッションハイジャックされる恐れがある

Amazonなどのサイトを開くと、今日はまだログインしていないのに、ログイン済みのページが表示されることがあります。これは以前ログインした際に、サイトからCookieで渡された「セッションID」（一時的な通行証のようなもの）をブラウザが保持していて、再びサイトを開いたときに提示することで、ログイン済みのユーザーとして扱われるからです。

ログインページだけが HTTPS になっているサイトだと、それ以外のページを HTTP で開いたとき、Cookie に Secure 属性が付いていなければ、この「セッションID」は暗号化されない状態で送信されてしまいます。では、誰でも使える Wi-Fi スポットに、スマートやタブレットを繋いだ状態で、HTTP のページを開いて「セッションID」が送られたらどうなるでしょう？

なんと同じ Wi-Fi につないでいる悪意の第三者によって、暗号化されていない「セッションID」を盗まれ、なりすましてサイトにログインされる恐れがあります。これが「セッションハイジャック」と呼ばれる攻撃です。

こうした攻撃を防ぐには、サイト全体を HTTPS にして、常に Cookie の「セッションID」を保護しておく必要があります。またログインなどが一切ない、公開情報しか載せていないサイトであっても、Wi-Fi スポットで見知らぬ誰かに「あの人はどんなサイトを見ているのだろう？」とデータを窺視されるリスクもあります。

例えば「妊活情報のブログを長時間読んでいた」「特定の病気について調べていた」など、どんなサイトを見ていたのか？という情報の中にも、人に知られたくないことはたくさんあります。サイトとの通信が HTTPS で保護されていれば、このような情報を盗み見られるリスクを低減できます。

5.4.3 相対的に検索順位が下がる

さらに、インターネット全体で HTTPS 化が進むことによって、HTTP のままでいるサイトに起きるデメリットもあります。

Google は「HTTPS everywhere」、つまり「(お問い合わせや会員登録といった一部のページに限らず) どこでも HTTPS！」を提唱しており、2014年8月の時点で既に、HTTPS に対応しているウェブサイトを検索ランキングで優遇する方針も発表^{*9}しています。

^{*9} Google ウェブマスター向け公式ブログ \[JA]: HTTPS をランキング シグナルに使用します https://webmaster-jp.googleblog.com/2014/08/https-as-ranking-signal.html

Googleのランキングアルゴリズムでは、「サイトがHTTPS化されているか」が指標のひとつとなっています。もちろんたくさんある指標の中のひとつで、他の指標ほどウェイトは大きくない、とされていますが、競合サイトのHTTPS化が進む中、自分のサイトだけHTTPのままでいると、相対的に検索順位が下がる可能性があります。

5.4.4 周りがHTTPSになるとリファラが取れなくなる

こちらはGoogleアナリティクスなどを使って、サイト流入元の情報を確認している方にとて、重要と思われるデメリットです。

自社のサイトがHTTPだと、HTTPSの他社サイトからリンクを踏んで飛んできた場合に、リファラ（利用者が直前に訪問していたサイトの情報）を取得することができません。HTTPのサイトでGoogleアナリティクスを使っている方は、実際にGoogleアナリティクスのページを開いて、集客の「参照元/メディア」を確認してみてください。アクセス元が「(direct) / (none)」と表示されて、どこから飛んできたのか分からぬものがありますか？その中には、ブラウザのブックマークや、メール内のリンクから飛んできた、本当に「直前に訪問していたサイトが存在しないもの」だけでなく、HTTPSのサイトから飛んできたアクセスも含まれています。

今後、周囲のサイトのHTTPS化が進んで、自社のサイトだけがHTTPで取り残されると、この「リファラが取得できる割合」はますます下がっていくことになります。自社のサイトをHTTPSにすれば、他社のHTTPSサイトから飛んできた場合でも、リファラを取得できるようになります。

5.5 HTTPS化すると得られるメリット

HTTPのままでいると起きるデメリットだけでなく、HTTPS化すると得られるメリットももちろんあります。

5.5.1 表示速度が上がる

かつては、HTTPSにすると、暗号化と復号のオーバーヘッドでHTTPのときよりもサイトが遅くなるというのが常識でした。しかし、サーバ側、クライアント側ともにCPUの性能が向上したことで、2020年現在、もはやオーバーヘッドは重要視するほどではなくなっています。^{*10}

^{*10} サーバの手前に設置して、暗号化や復号だけを行なう専用機器の「SSLアクセラレータ」を使う話も、最近はほぼ聞かなくなりました

それだけでなく、HTTPS にすることで HTTP/2 というプロトコルを利用できるようになります。HTTP/2 では複数のリクエストを同時平行で処理できるため、むしろ表示の速度が速くなるケースもあります。

さらに TLS1.3 になると、TLS1.2 に比べてハンドシェイクに要するやりとりの回数や所要時間が大幅に短くなったため、さらなる速度改善が見込まれます。

5.5.2 SameSite の変更に対応できる

2020年2月にリリースされたChromeのバージョン80^{*11}から、CookieのSameSiteの設定が従来より厳しくなり、今まで設定なしで使えていたクロスサイトCookieが、デフォルトでは使えないよう順次変更されていくことが発表されました。

ですが「SameSite=None; Secure」の設定をすれば、クロスサイトCookieは引き続き利用できます。そしてSecure属性を追加するには、HTTPSでの接続が必須です。

このように、HTTPSであればSameSiteの変更にも対応できる、というメリットがあります。

5.5.3 重要情報のアタリが付けにくくなる

これは自分のサイトだけでなく、インターネット上のサイト全体がHTTPSに対応したときに得られるメリットです。

流れしていくデータの殆どが平文な中で、たまに暗号化されたデータがやってくると、データを窺視していた悪意の第三者は「暗号化されているということは、あれは重要な情報だな！」とアタリを付けて狙うことができます。ですが、すべてのサイトがHTTPSになって、流れてくるデータがすべて暗号化されるようになれば、どれが重要なデータなのかアタリが付けにくくなります。大事なものも大事でないものもすべて同じ頑丈なアッシュケーズにしまって運べば、泥棒がどのアッシュケーズを狙えばいいのか分からなくなります。木は森に隠せ、という戦法です。

このように、HTTPのままだと起きるデメリット、HTTPSにすると得られるメリット、その両方があるのでどんなサイトもHTTPS化する意味がある、ということですね。

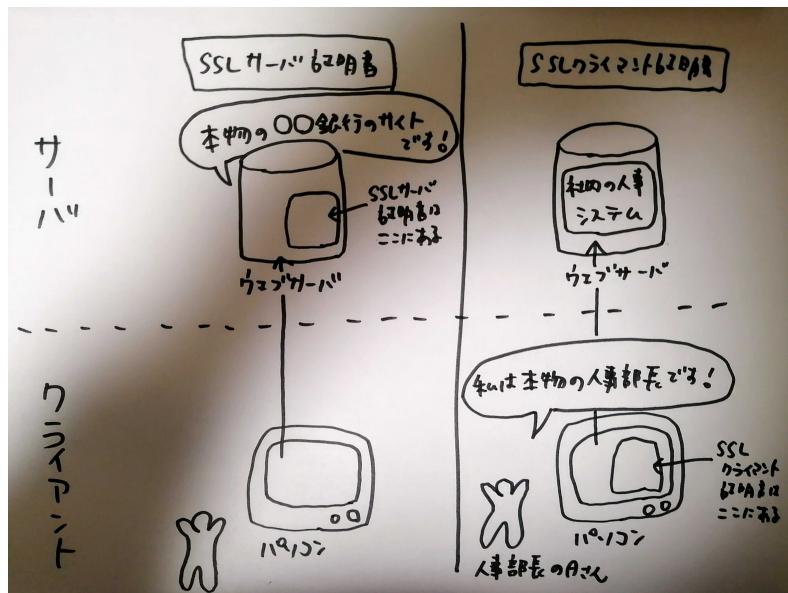
5.6 SSL 証明書とは

続いて、HTTPS化するため必要なSSL証明書について学んでいきましょう。

^{*11} <https://developers-jp.googleblog.com/2019/11/cookie-samesitenone-secure.html>,
<https://developers-jp.googleblog.com/2020/02/2020-2-samesite-cookie.html>

5.6.1 SSL サーバ証明書と SSL クライアント証明書

皆さんがさっそく取得、設置した「SSL 証明書」ですが、実は SSL サーバ証明書と SSL クライアント証明書の2種類（図5.3）があります。ざっくり言うとサーバの身元を証明するのが SSL サーバ証明書で、クライアントの身元を証明するのが SSL クライアント証明書です。

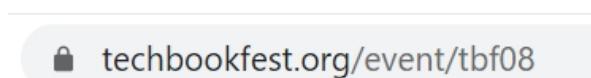


▲図 5.3 SSL サーバ証明書と SSL クライアント証明書

単に「SSL 証明書」という略称で呼んだ場合は、「SSL サーバ証明書」のことを指すことが殆どです。本著でも SSL サーバ証明書のことを指して、SSL 証明書という言葉を使用しています。

5.6.2 SSL 証明書はどんな場面で使われている？

SSL 証明書はどんな場面で使われているのでしょうか？ SSL 証明書は、あなたがブラウザで「https://」から始まる URL のサイトを開いて、次のようなマーク（図5.4、図5.5）が表示されているときに使われています。



techbookfest.org/event/tbf08

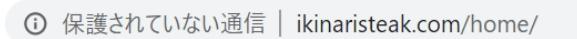
▲図 5.4 Chrome で HTTPS のサイトを開いたとき



https://techbookfest.org/event/tbf08

▲図 5.5 Firefox で HTTPS のサイトを開いたとき

逆に「http://」から始まる URL のサイトを開いて、次のようなマーク（図 5.6、図 5.7）が表示されているときは使われていません。



① 保護されていない通信 | ikinaristeak.com/home/

▲図 5.6 Chrome で HTTP のサイトを開いたとき



ikinaristeak.com/home/

▲図 5.7 Firefox で HTTP のサイトを開いたとき

5.7 SSL 証明書は異なる 3 つの仕事をしている

「SSL 証明書は https://から始まるページで使われている」ということが分かりました。では https://から始まるページにおいて、SSL 証明書は一体何をしているのでしょうか。

実は、SSL 証明書は異なる 3 つの仕事をしています。たとえば、あなたが銀行振込をするためにイグザンブル銀行のウェブサイト (<https://bank.example.com/>) に接続したとき、SSL 証明書は次のような 3 つの仕事をします。

- なりすましを防ぐ

- ・「bank.example.comさん、ページを見せて」というリクエストに対して、レスポンスでページを返してきたのが、本当にbank.example.comであることを認証する
- ・データの改ざんを防ぐ
- ・イグザンブル銀行のウェブサイトで入力した振込先の情報が、サーバに届くまでに他の振込先に書き換えられていないことを確認する
- ・リクエストに対して返ってきた残高のページが、クライアントまでの途中経路で改ざんされていないことを確認する
- ・情報の漏洩を防ぐ
- ・サーバへ送信したIDやパスワード、振込先の情報などが第三者に見られないよう暗号化して保護する
- ・入出金の明細がサーバからクライアントへ届くまでの間に、第三者に見られないよう暗号化して保護する

厳密には、3つめの暗号化はSSL証明書そのものの働きではなく、暗号化に必要な鍵交換を行なう過程でSSL証明書が使われます。

5.7.1 HTTPS の実際の流れ

ではHTTPSの全体を流れを掴むため、あなたがさっき作ったばかりの自分のサイト(<https://ssl.自分のドメイン名/>)を開いたときの、ざっくりとした手順を追いかけてみましょう。あなたのブラウザが「クライアント」で、Oracle Cloud上で立てたHTTPSのサイトが動いているのが「サーバ」です。

認証を行なう

先にRSA^{*12}による認証を行ないます。

1. ブラウザでHTTPSのサイト(<https://ssl.自分のドメイン名/>)を開く
2. クライアントからOracle Cloudのウェブサーバへリクエストを投げる
3. サーバがリクエストを受け取る
4. サーバ内にあるFujiSSLのSSL証明書をレスポンスで返す* この「SSL証明書」は次の2つを指す
5. SSL証明書本体(公開鍵を含む)

^{*12}もともとは認証だけでなく鍵交換もRSAで行なわれていましたが、RSAによる鍵交換は、一度秘密鍵が盗まれてしまうと、過去のやりとりもさかのぼって全ての暗号データを復号可能という問題がありました。そのためTLS1.3ではRSA鍵交換は廃止されています

6. 認証局による署名（SSL 証明書本体のハッシュ値を認証局の秘密鍵で暗号化^{*13}したもの）
7. クライアント側で SSL 証明書を受け取って次の 3 つを行なう
8. ブラウザのトラストアンカー（信頼する証明書）に含まれている認証局によって発行された SSL 証明書なのか確認* これで渡された SSL 証明書を信頼してよいことが分かる
9. 認証局による署名を、ブラウザのトラストアンカー（信頼する証明書）に含まれる公開鍵で復号して、証明書本体のハッシュ値を取り出す
10. ハッシュ関数で SSL 証明書本体のハッシュ値を出力
11. 取り出したハッシュ値と、自分で出力したハッシュ値を突き合わせて同一であることを確認する* これで渡された SSL 証明書本体が改ざんされていないことが分かる
12. SSL 証明書本体の SAN に記載されている FQDN^{*14}と、リクエスト先の FQDN (bank.example.com) が同一であることを確認* これでレスポンスを返してきた相手がなりすましてないことが分かる

DH 鍵交換による暗号化通信を行なう

認証が終わると、続けて DH 鍵交換^{*15}を行ないます。

1. サーバは DH 公開鍵交換のために使い捨ての鍵ペア（秘密鍵・公開鍵）を作る* この鍵ペアは、SSL 証明書を取得するときに作った鍵ペア（秘密鍵・公開鍵）とはまた別物* 以後この鍵ペアを「サーバの DH 用の秘密鍵・公開鍵」と呼ぶ
2. クライアントも DH 公開鍵交換のために使い捨ての鍵ペア（秘密鍵・公開鍵）を作る* 以後この鍵ペアを「クライアントの DH 用の秘密鍵・公開鍵」と呼ぶ
3. サーバは SSL 証明書用の秘密鍵で、サーバの DH 用の公開鍵を暗号化してクライアント側に渡す
4. クライアントは「認証」で受け取った SSL 証明書本体に含まれていた「SSL 証明書用の公開鍵」で、サーバの DH 用の公開鍵を復号する
5. クライアントは自分が作った DH 用の公開鍵を、サーバの DH 用の公開鍵で暗号

*¹³ ここは実態としては「秘密鍵で復号」らしい。まだ暗号化していないものを復号するイメージを説明できなかったので

*¹⁴ FQDN は Fully Qualified Domain Name の略で、日本語だと完全修飾ドメイン名。example.co.jp というドメイン名があったとき、個々の example や co や jp や example.co などは相対ドメイン名で、example.co.jp が FQDN です

*¹⁵ Diffie-Hellman 鍵交換。TLS1.3 からは

化してサーバに送る

6. サーバは、受け取ったクライアントの DH 用の公開鍵を、サーバの DH 用の秘密鍵で復号する
7. これでクライアントとサーバはお互いに相手の DH 用の公開鍵を持っている状態になる
8. サーバの DH 用の公開鍵と、クライアントの DH 用の公開鍵のセットを、共通鍵の素（プリマスターシークレット）と呼ぶ
9. クライアントサーバは、それぞれプリマスターシークレットを素にして共通鍵を作る
10. 以降、同一セッションの間は、双方この共通鍵で暗号化および復号してデータをやりとりする

このように RSA による認証と、DH 鍵交換の組み合わせで、HTTPS の暗号化通信が行なわれています。

5.7.2 ウェブページは1往復で表示されるわけじゃない

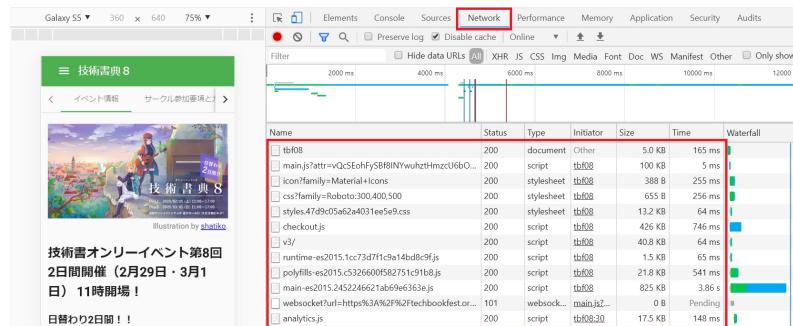
ところで1つのウェブページを表示するとき、クライアント（あなたのブラウザ）とサーバのやりとりは、「トップページをください」「はい完成品をどうぞ」の一往復だけではありません。次のように、何往復ものリクエストとレスポンスでペーツを揃えていくてページが表示されます。

- 「techbookfest.org さん、tbf08 のページをください」「はい、HTML をどうぞ」
- 「techbookfest.org さん、main.js をください」「はい、main.js をどうぞ」
- 「techbookfest.org さん、styles.css をください」「はい、styles.css をどうぞ」
- 「techbookfest.org さん、top.jpg をください」「はい、top.jpg をどうぞ」

Chrome のメニューから【その他のツール】の【デベロッパーツール】を開いて、[Network] タブを選択した状態で、たとえば技術書典8のサイト^{*16}を開くと、次のようにたくさんの行が表示（図5.8）されます。2019年2月時点、このページは54往復のリクエストとレスポンスで表示されており、この1行1行が「○○をください」「はい、どうぞ」という、リクエストとレスポンスの往復を表しているのです。

^{*16} <https://techbookfest.org/event/tbf08>

5.7 SSL 証明書は異なる 3 つの仕事をしている

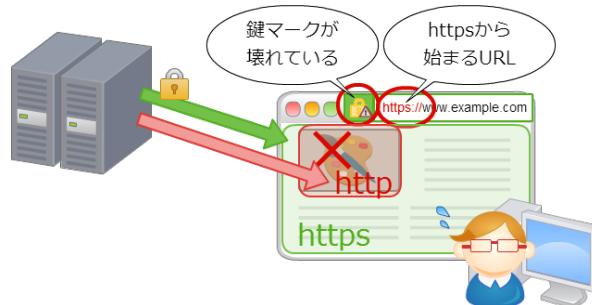


▲図 5.8 54 往復のリクエストとレスポンスで表示された技術書典 8 のページ

5.7.3 HTTP の混在コンテンツ（画像や CSS）があるとブロックされる

「HTTPS から始まる URL を開いたときに、鍵マークではなく i マークが表示される」という現象には、この「ページの表示は一往復じゃない」という部分が大きく関係しています。

通常、ブラウザで `https://` から始まる URL を開くと、前述のとおり、リクエストしたページは HTTPS で暗号化された状態で届きます。しかし取得したページの HTML で、「``」のように、絶対パスで画像を指定する `` タグが含まれていた場合、その画像ファイルは暗号化されていない HTTP で送られてきます。（図 5.9）



▲図 5.9 混在コンテンツ（mixed content）はエラーが出る

つまり、せっかくウェブページを HTTPS にしても、そのウェブページの HTML の中で、CSS や画像ファイルを `http://` から始まる形式にしていたり、YouTube などのコン

テンツを `http://`から始まる形式で埋め込んでいると、ページが表示されるまでのたくさんの往復の中で、一部が HTTP になってしまっているので、ブラウザが鍵マークの代わりに i マークを表示して、[このサイトへの接続は完全には保護されていません] と警告(図 5.10)を出すのです。



▲図 5.10 [このサイトへの接続は完全には保護されていません] と警告が出る

この問題は混在コンテンツ (mixed content) と呼ばれています。2020年3月にリリース予定の Chrome のバージョン 81^{*17}からは、この混在コンテンツが存在した場合、対象のプロトコルが HTTP から HTTPS へ自動的に変更され、さらに HTTPS での読み込みに失敗すると、そのリソース (`http://`で始まる形式にしていた CSS や画像、埋め込みコンテンツ) がブロックされる変更が予定されています。

混在コンテンツを直すには、たとえば「」のようになっていたタグを「」のようにパスの部分だけにします。これならページを HTTP で開いたときは画像も HTTP で、ページを HTTPS で開いたときは画像も HTTPS で表示されるため、混在コンテンツにはなりません。

あるいは「☒ 画像はページとは別のドメイン名なので、パスだけでなくド

*17 <https://developers-jp.googleblog.com/2019/11/https.html>

メイン名から指定しなければいけない」という場合は、タグを「」のようにプロトコルを省略して書くことで、先ほどと同じように、ページを HTTP で開いたときは画像も HTTP で、ページを HTTPS で開いたときは画像も HTTPS で表示されます。このようにして混在コンテンツを解消してやれば、きちんと鍵マークが表示されるようになります。

ちないに「Google HTML/CSS Style Guide^{*18}」では、このプロトコルを省略する書き方は非推奨とされています。ページが HTTP か HTTPS かに関わらず、画像は HTTPS で表示して構わない、という場合は、「」のように指定するのがいいでしょう。

5.7.4 正当性を証明する中間 CA 証明書

ではここで、サイバートラストが提供する「SSL 証明書の設定確認ツール」^{*19}を使って、あなたが作ったサイトの SSL 証明書を確認（図 5.11）してみましょう。[FQDN1] に [ssl.自分のドメイン名] を入力し、[設定を確認する] をクリックします。



▲図 5.11 [ssl.自分のドメイン名] を入力して [設定を確認する] をクリック

すると「証明書は正しく設定されています。」というメッセージと共に、証明書の階層が表示（図 5.12）されました。

*18 <https://google.github.io/styleguide/htmlcssguide.html#Protocol>

*19 https://sstool.cybertrust.ne.jp/support_tool/index01.php

第5章 SSL/TLSについて学ぼう



▲図 5.12 証明書の階層が表示された

[中間 CA 証明書 1] と [中間 CA 証明書 2] で、それぞれ [詳細情報を表示する] をクリックすると、詳細が表示されます。いろいろなことが記載されていますが、要約すれば次の表に挙げたことが書かれています。

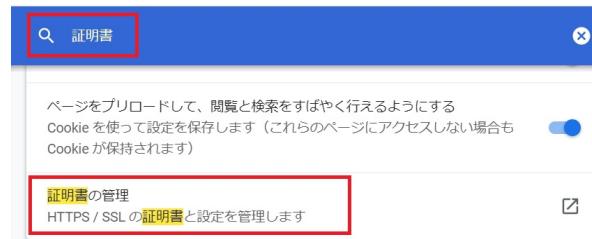
- SSL サーバ証明書
 - [ssl.startdns.fun] というサイトの運営者が、確かにそのドメイン名を管轄していることを、[SECOM Trust Systems CO.,LTD.] が証明する
- 中間 CA 証明書 1
 - SSL サーバ証明書を発行した [SECOM Trust Systems CO.,LTD.] が正当な認証局であることを、[Security Communication RootCA2] が証明する
- 中間 CA 証明書 2
 - 中間 CA 証明書 1 を発行した [Security Communication RootCA2] が正当な認証局であることを、[Security Communication RootCA1] が証明する

信頼の鎖ですね。[ssl.startdns.fun] というサイトの運営者を仮に A さんとすると、A さんがドメイン名の持ち主であることを B さんが、B さんの正当性を C さんが、C さんの正当性を D さんが証明しています。

SSL 証明書と中間 CA 証明書 1、2 は [startssl.crt] というファイル名でサーバに設置されており、HTTPS でサイトを開いたときにサーバからクライアント（ブラウザ）へ渡されました。ですがあなたは身も知らない D さんこと [Security Communication RootCA1] を信頼できますか？ D さんの正当性はいったい誰が証明するのでしょうか？

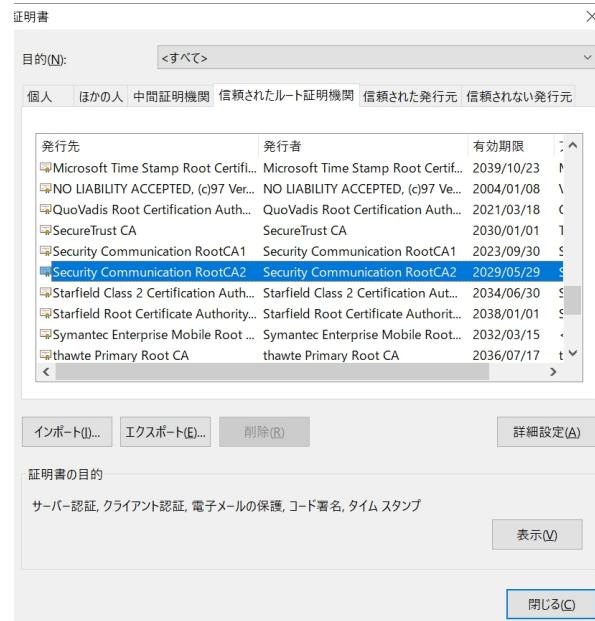
5.7.5 ルート証明書はトラストストアにある

実は、中間 CA 証明書 2 を発行した D さんこと [Security Communication RootCA1] の正当性を証明する「ルート証明書」は、皆さんのが使っているブラウザに最初から入っています。Chrome で [設定] を開いたら [証明書] で検索（図 5.13）して、[証明書の管理] をクリックします。



▲図 5.13 [証明書の管理] をクリック

[信頼されたルート認証機関] のタブをクリック（図 5.14）すると、そこに D さんこと [Security Communication RootCA1] の正当性を証明する「ルート証明書」がありました。ちなみにルート証明書の〔発行先〕と〔発行者〕を見ると、どちらも同じ [Security Communication RootCA1] になっています。これは D さんの正当性は D さん自身が証明し、ブラウザが D さんのルート証明書を「信頼できるルート証明書である」と判断して格納場所（ルートトラストストア）に入れているので、信頼の連鎖が繋がった、ということです。



▲図 5.14 [信頼されたルート認証機関] にルート証明書があった

このように SSL 証明書と中間 CA 証明書がウェブサーバにあり、信頼の起点となるルート証明書がブラウザにあることで、SSL 証明書は成り立っています。SSL 証明書ってこんな仕組みになっていたんですね。

5.8 SSL 証明書はどうしてあんなに値段に差があるの？

SSL 証明書の仕組みが分かったところで、SSL 証明書のあるあるな疑問、「どうして SSL 証明書はあんなに値段に差があるの？」について少しお話しておきましょう。

SSL 証明書で検索してみると、シマンテックは 219,000 円、サイバートラストは 75,000 円、そしてサイフにやさしい SSL 証明書こと RapidSSL は 2,600 円でした（執筆当時）。この価格差は驚きますよね。

なぜこんなに価格差があるのでしょうか。シマンテックはブランド代が含まれるので高いのでしょうか？ あるいは RapidSSL が企業努力の塊なのでしょうか？ それとも同じ「SSL 証明書」という名前でも、RapidSSL の SSL 証明書は中身が何か違うのでしょうか？

5.9 同じ「SSL 証明書」という名前でも 3 つの種類がある

結論から言うと違います。

SSL 証明書の役割は次の 2 つであることを説明してきました。

ウェブサイトで送受信する情報を暗号化することウェブサイト運営者の身元を証明することしかし、RapidSSL が発行しているような安い SSL 証明書は、ウェブサイト運営者の身元証明をせず、「情報の暗号化」だけしか行いません。

「え、それって SSL 証明書だって言えるの？」と思うかもしれません、言えます。なぜなら、SSL サーバ証明書には実は 3 つの種類があるからです。

「SSL 証明書」とひとくちに言っても、その実態は 3 種類に分かれています。分かりやすくいうと「高い」「普通」「安い」の 3 種類で、それぞれ「EV 証明書」「 OV 証明書」「 DV 証明書」という名前です。

5.9.1 3 つの違いは何か？

種類 何を証明してくれる？ 商品例 EV 証明書 (Extended Validation) ウェブサイト運営者の身元をより厳格に書類と電話で確認して証明 ・ サイバートラストの SureServer EV ・ シマンテックのセキュア・サーバ ID EV OV 証明書 (Organization Validation) ウェブサイト運営者の身元をメールと電話で確認して証明 ・ サイバートラストの SureServer

- ・シマンテックのセキュア・サーバ ID DV 証明書（Domain Validation） そのドメインの使用権があることを証明 ・RapidSSL RapidSSLのような DV 証明書は、SSL 証明書という名前で呼ばれていますが「ウェブサイト運営者の身元証明」は一切行いません。

5.9.2 DV 証明書

ドメインの所有者を「Whois」と呼ばれるドメインの登録者情報で確認し、そこに書いてある所有者のメールアドレスに対して「このドメインの SSL 証明書を発行していいですか？」と確認してくるだけです。ですから、ドメイン所有者が発行承認ボタンを押したら、それだけですぐに発行されます。

このように DV 証明書は、「 ドメインの使用権があること」の確認と証明をするだけで、「 誰がそのウェブサイトを運営しているのか？」という身元確認及び身元証明はしてくれません。したがって DV 証明書は、先ほどの EXAMPLE 銀行のように「身元の証明をしたい」ケースでは使う意味がありません。HTTPS で暗号化はしたいけれど、身元証明をする必要度はあまりない、開発用のテスト環境等で使用されることが多いです。

5.9.3 EV 証明書と OV 証明書

DV 証明書の役割は分かりました。では残りの 2 つ、「 高い = EV 証明書」と「普通 = OV 証明書」の違いは何なのでしょうか。先ほど例として挙げた株式会社イグザンブルのコーポレートサイトでは、「 普通」の OV 証明書を採用しています。

EV 証明書と OV 証明書は、DV 証明書と違って、SSL 証明書の 2 つの役割をきちんと果たします。

しかし OV 証明書は、ブラウザの鍵マークをクリックして証明書を開き、その中の証明書情報を確認しないと、サイト運営者の名前が表示されません。分かりにくくと思いますので、実際の画面を見てみましょう。例えば Firefox で見たとき、株式会社イグザンブルのコーポレートサイトのお問い合わせ画面は次のようになります。

驚かれるかもしれません、OV 証明書の場合、ここにウェブサイトの運営者名は表示されず、「 このサイトの運営者は不明です」になってしまいます。なぜならば「ウェブサイト運営者の身元をメールと電話で確認しただけで、確認度合いが低いため、身元証明がいまいち信用できない」とブラウザ（Firefox）が思っているからです。

鍵のマークをクリックした後に、「 詳細を表示...」をクリックし、さらに「証明書を表示...」をクリックすると、ここで初めて運営者名として「EXAMPLE Corporation」という名前が出てきます。EV 証明書のように、「 ウェブサイト運営者の身元を書類と電話でより厳格にチェックして証明」されたものでないと、ここには運営者の名前は出ないの

5.9 同じ「SSL 証明書」という名前でも 3 つの種類がある

です。

Firefox の証明書詳細情報確認画面

画像

これは結構重要な問題です。つまり一見しただけなら、身元を証明しない DV 証明書と、身元を証明する OV 証明書は区別がつかないのです。先ほどの EXAMPLE 銀行の例に戻ってみましょう。EXAMPLE 銀行の広報担当が身元証明をする OV 証明書を取得して、詐欺師が身元を証明しない DV 証明書を取得した場合、次のようにになります。

OV 証明書と DV 証明書はぱっと見ただけでは区別がつかない

画像

もちろん、証明書の情報をよく確認してもらえば、片方が身元証明されていないことは分かるのですが、そんなことをするエンドユーザは滅多にいません。

そこで、ぱっと見ただけで「偽物と区別がつくようにならせる」「成りすましを防ぎたい」というときは、DV 証明書でも OV 証明書でもなく、EV 証明書を使う必要があります。実際、三井住友銀行や三菱東京 UFJ 銀行を始めとする国内のネット銀行は、ほとんどが EV 証明書を採用しています。

シマンテックの EV 証明書を採用している三菱東京 UFJ 銀行のサイト

画像

またネットショップのように、クレジットカード情報を入力するサイトでも、EV 証明書を採用するところが増えました。例えば「山田養蜂場」というはちみつや自然食品のオンライン販売をしているサイトでは、サイバートラストの EV 証明書を使っているため、URL の左側にサイトの運営者名が日本語で出ています。また鍵マークをクリックすると、社名に加えて住所も表示されるため、どこのだれが運営しているサイトなのかがすぐに分かります。

EV 証明書を採用している山田養蜂場のサイト

画像

ネット銀行やネットショップなど、偽物が出やすく、かつ偽物による被害が出た場合にダメージが大きいサイトでは、多少値段が高くても EV 証明書を使う意味があるということなのです。

このように「SSL 証明書」という 1 つの名前でも、その中で「DV 証明書」「 OV 証明書」「 EV 証明書」の 3 種類に分かれています。高い SSL 証明書と安い SSL 証明書がある理由が納得できましたか？

5.9.4 さよならグリーンバー

2019年9月にリリースされたChromeバージョン77、そして翌月10月にリリースされたFirefoxバージョン70からは、グリーンバーは表示されなくなりました。

5.9.5 ブラウザベンダーによるEV証明書の扱いの変化

5.10 その他の証明書

5.10.1 中間証明書

5.10.2 クロスルート証明書

5.11 どの証明書を買えばいい？

5.11.1 ワイルドカード証明書

5.11.2 wwwありにリダイレクトしたいだけなのにwwwなしの証明書もいるの？

5.11.3 コモンネームが*.example.comの証明書はexample.comで使える？

5.11.4 SANs

5.11.5 Let'sEncrypt

5.12 CDNと証明書

5.12.1 CDNを使ったら古い端末でサイトが見られなくなった

5.12.2 同じサーバで複数サイトをHTTPS化したら古い端末で別サイトが表示された

5.12.3 SNI Server Name Indication

HTTPSで使われるTLSプロトコルでは、接続したいホスト名をクライアント側からサーバに伝えるためにSNI(Server Name Indication)のTLS拡張が必要となります。

ただし SNI は、1 つの IP アドレスを複数のバーチャルホストで共用するため、HTTPS で使用した場合、SNI 非対応のクライアントではデフォルトのホストが応答します。2019 年現在、SNI 非対応端末を「対象端末」としているサービスはあまり多くないかもしれません。

あとがき

好きな技術の本を書くのは、とても楽しいものです。「誰に頼まれた訳でもないのになぜ書くの？」と自分に問いかけると、そこには、「これを書いたら、きっと誰かの助けになるはずだ」という祈りのような希望があります。

祈りのはかなさに対して、自覚すらない悪意はとても強いです。第三者から気軽にぶつけられた石の痛みに呆然としながら、「もう書くのやめようかな」と思うときもあります。

以下は、そんな筆者に対して、同僚であり友人でもある Marshall が送ってくれたメッセージです。何かを書いたり、作ったりして、誰かの役に立とうとしている人、みんなの胸に灯りをともすようなメッセージだったので、あとがきに載せる許可をもらいました。

快く承諾してくれた Marshall に感謝します。

To be honest, people can find **ANY** information on the internet. Everything I learned at my university I could have learned from the internet--I still went to university.

There is a lot of information floating around online. There is a lot of information about marketing, but I still have a ton of books at home on how to become a better marketer.

People can take pictures with their smartphone--that doesn't mean people who draw should stop drawing or people who paint should stop painting.

Keep writing!

I've written a lot of articles online with the intention of helping people. People will leave comments like "What a stupid article! You think people don't already know how to do this? Why waste your time writing this!?"

But my answer is because I know my article helped someone and that makes me happy. Not everyone knows everything and if I can share my knowledge and help at least one person, then it is worth it for me.

The point I'm trying to make is no matter what you do there is someone who will try to make you feel bad about it. Keep doing it anyway.

数ある技術書の中から「SSLをはじめよう」を手に取ってくださったあなたに感謝します。

2020年3月
mochikoAsTech

PDF版のダウンロード

本著（紙の書籍）をお買い上げいただいた方は、下記のURLからPDF版を無料でダウンロードできます。

- ダウンロード URL : <https://mochikoastech.booth.pm/items/xxxxxx>
- パスワード : **xxxxxx**

Special Thanks:

- Gunnell Marshall

レビュアー

- Takeshi Matsuba
- Mari Kubota

参考文献・ウェブサイト

- プロフェッショナル SSL/TLS Ivan Ristić、齋藤孝道（監訳）
 - <https://www.lambdanote.com/products/tls>
- プロフェッショナル IPv6 小川晃通
 - <https://www.lambdanote.com/products/ipv6>
- 食べる！SSL！—HTTPS環境構築から始めるSSL入門 小島拓也、中嶋亜美、吉原恵美、中塚淳
 - <https://www.amazon.co.jp/dp/B00PHC4480>
- SSL/TLSの基本 - Qiita
 - https://qiita.com/angel_p_57/items/446130934b425d90f89d

-
- 【図解】初心者も分かる”公開鍵/秘密鍵”の仕組み～公開鍵暗号方式の身近で具体的な利用例やメリット～ | SE の道標
 - <https://milestone-of-se.nesuke.com/sv-advanced/digicert/public-private-key/>
 - 「電子署名=『秘密鍵で暗号化』」という良くある誤解の話 - Qiita
 - https://qiita.com/angel_p_57/items/d7ffb9ec13b4dde3357d

著者紹介

mochiko / @mochikoAsTech

元 Web 制作会社のシステムエンジニア。技術書典で出した本がきっかけで、テクニカルライターの仕事を始めた。モバイルサイトのエンジニア、SIer とソーシャルゲームの広報を経て、2013 年よりサーバホスティングサービスの構築と運用を担当したのち、再び Web アプリケーションエンジニアとしてシステム開発に従事。「分からない気持ち」に寄り添える技術者になれるように日々奮闘中。技術書典 4,5,6 で頒布した「DNS をはじめよう」「AWS をはじめよう」「技術をつたえるテクニック」「技術同人誌を書いたあなたへ」は累計で 7,800 冊を突破。

- <https://twitter.com/mochikoAsTech>
- <https://mochikoastech.booth.pm/>
- <https://note.mu/mochikoastech>
- <https://mochikoastech.hatenablog.com/>

Hikaru Wakamatsu

表紙デザインを担当。

Shinya Nagashio

挿絵デザインを担当。

SSLをはじめよう

「なんとなく」から「ちゃんとわかる！」へ

2020-02-29/2020-03-01 技術書典 8 初版

著 者 mochikoAsTech

デザイン Hikaru Wakamatsu / Shinya Nagashio

発行所 mochikoAsTech

印刷所 日光企画

(C) 2020 mochikoAsTech