知らないと困る？！ 認証局とHTTPSの最新技術動向

運用の観点から見たTLSプロトコルの動き

大津 繁樹
ヤフー株式会社

Internet Week 2017
2017年11月29日

自己紹介

- ヤフー株式会社: CDNチーム所属、IETF標準化参加、黒帯（ネットワーク・セキュリティ）

- Node.js コミッター: tls,crypto周りの実装、セキュリティチーム所属 (ヤフー株式会社 OSSデベロッパー認定者)

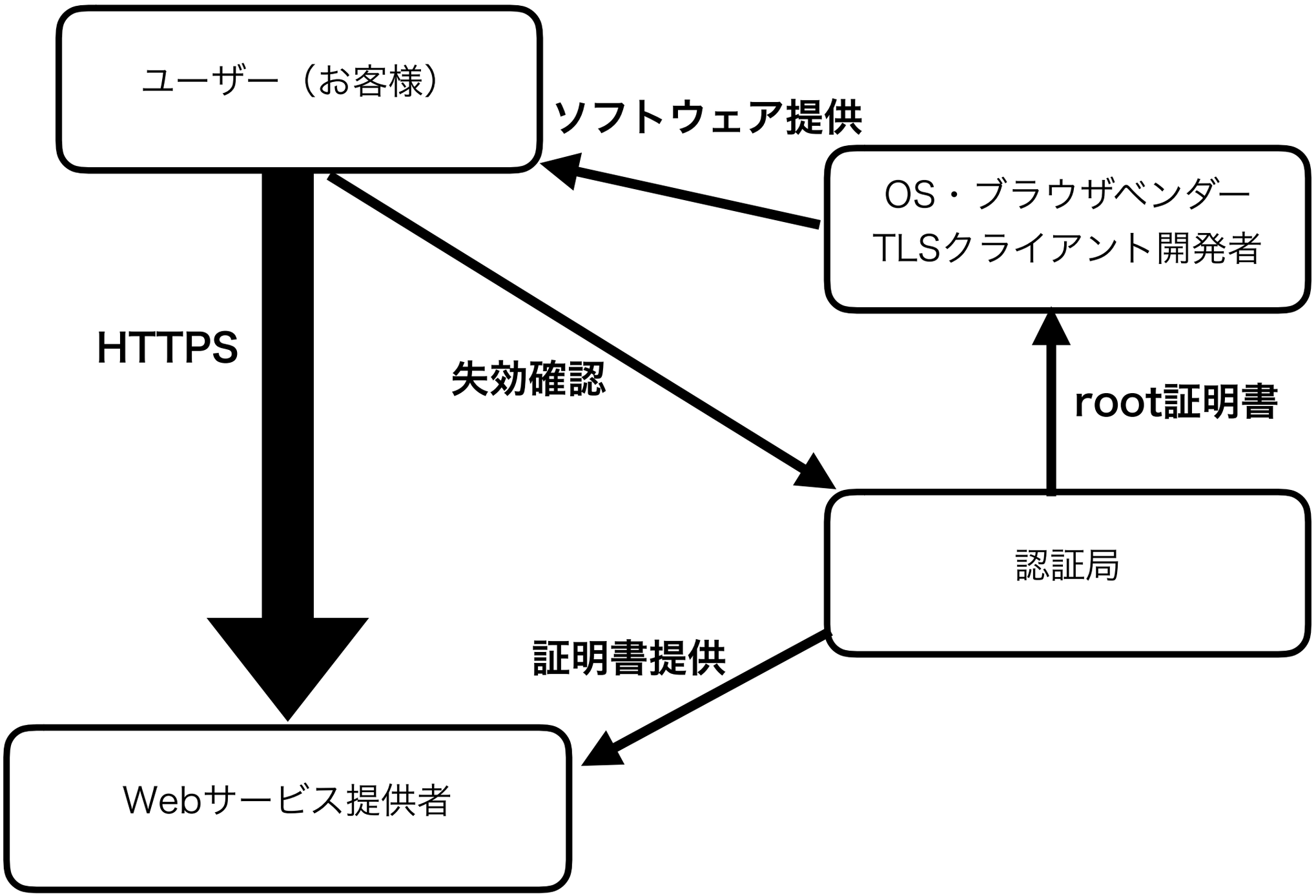
本日の内容

今後のHTTPSサービス運用で知っておくべきTLSプロトコルの現状の課題と最新動向について解説します。こんな疑問にお答えします。

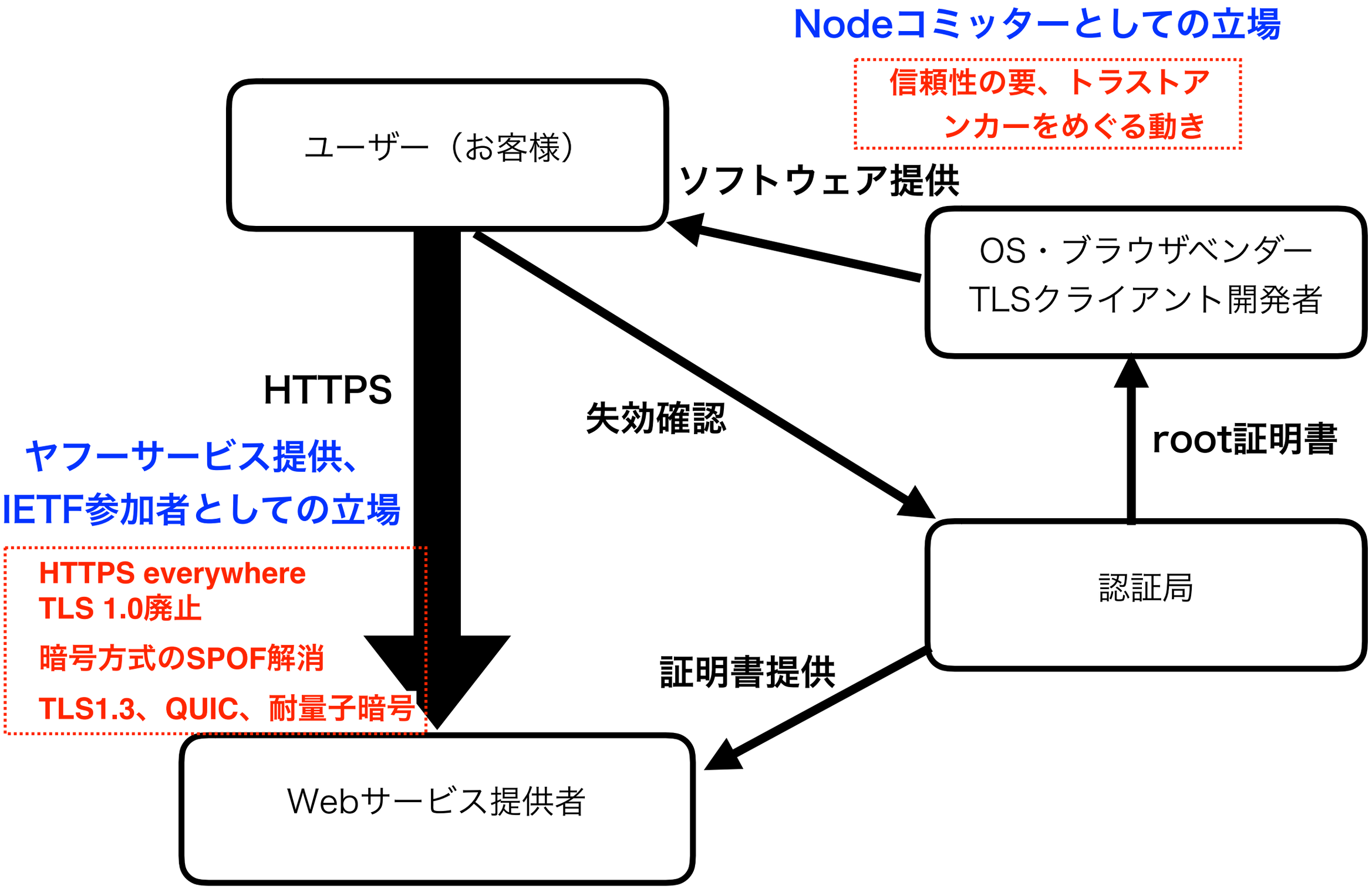
- なぜ全部HTTPSにしないといけないの？
- うちシマンテックの証明書使っているけど、どうなっちゃうの？
- うちのHTTPSサーバ、このままの設定で放おっておいて大丈夫？

1. どうしてHTTPSにしないといけないのか。
2. 信頼性の要、トラストアンカーをめぐる動き(認証局対ブラウザベンダー)。
3. 今後TLS1.0をどうしたらいいのか。
4. 暗号方式のSPOF解消に向けて(非NIST暗号の特徴)。
5. TLS1.3でどう変わるのか。
6. QUIC、耐量子暗号を見据えて。

HTTPS everywhere 時代のステークホルダー



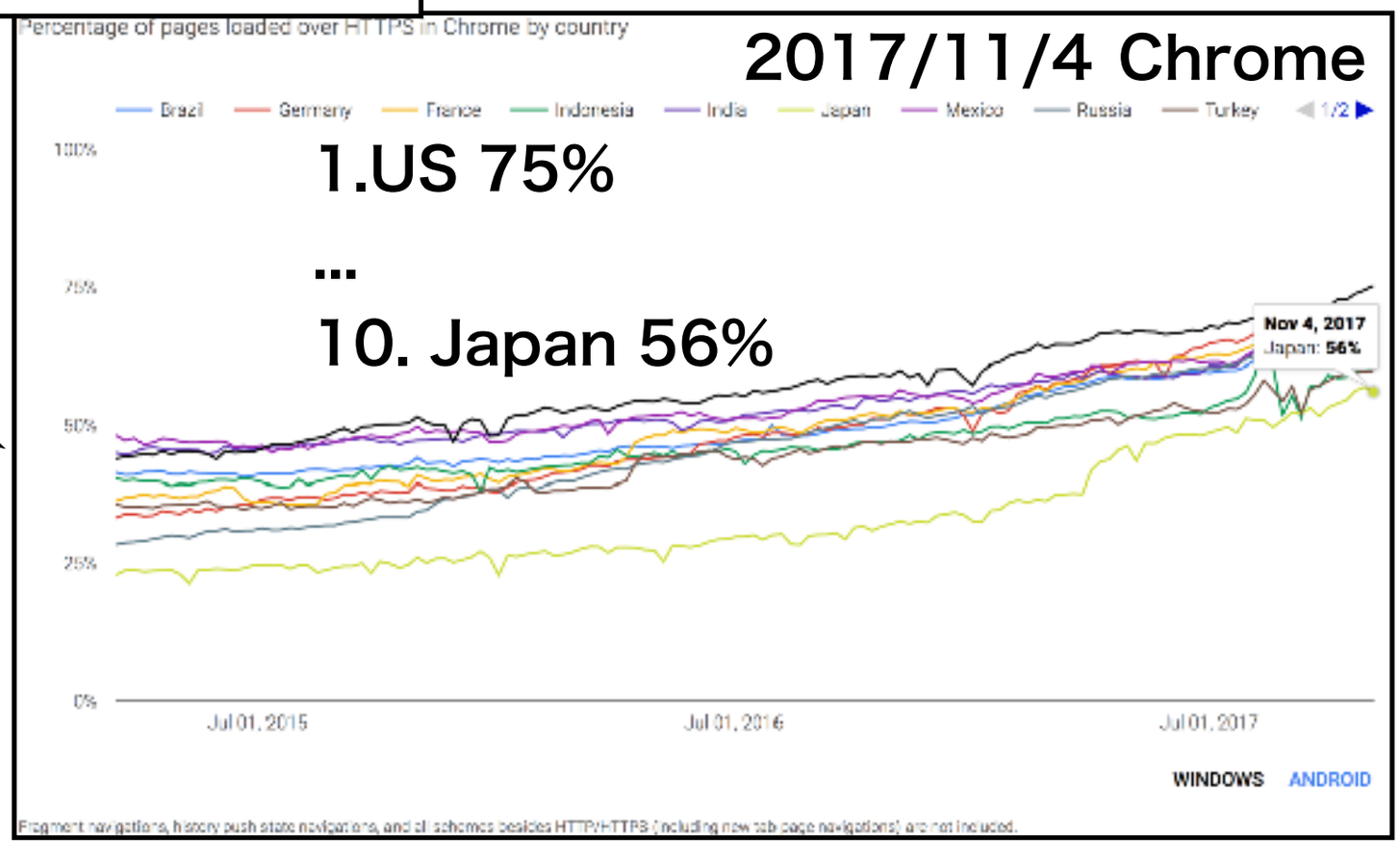
２つのポジションを使い分けして話をします



HTTPS Everywhere

HTTPSの導入状況(読み込みページ割合)

日本は56%
10カ国中最低



https://letsencrypt.org/stats/#percent-pageloads

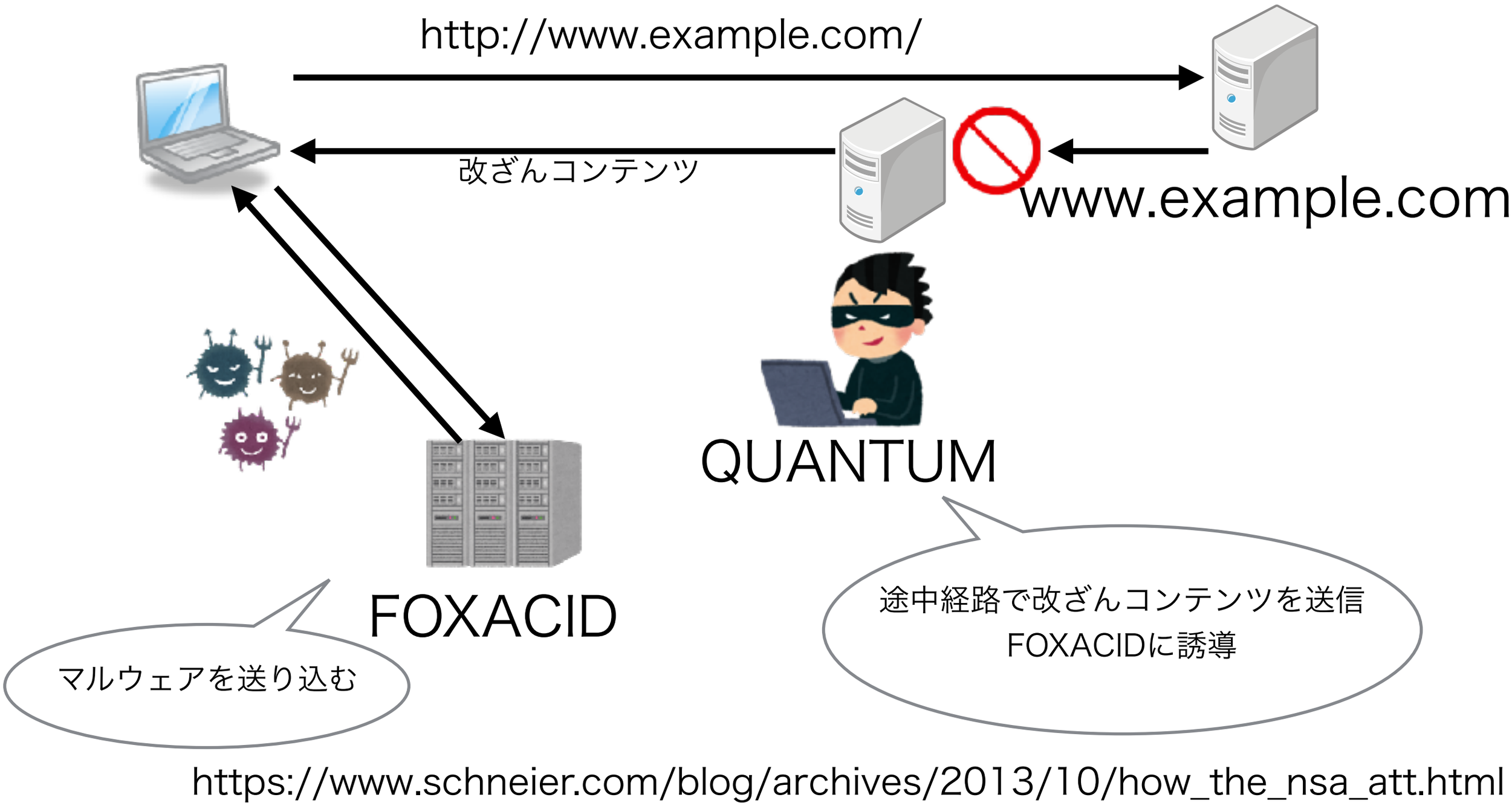
https://transparencyreport.google.com/https/overview

なぜ全部HTTPSにしないといけないの？

2013年6月： エドワード・スノーデン事件

- NSA, CGHQによる国家レベルの広範囲な盗聴行為が明らかに
- 通信キャリアと協力し、DC内やインターネットバックボーンに盗聴・改ざんの仕掛けを配置
- 平文通信をMiTMで改ざんし未知のマルウェアを感染させる
- 暗号の標準化仕様にバックドアを仕掛けているとの話も(後述)

NSAによるサイバー攻撃の一例



https://www.schneier.com/blog/archives/2013/10/how_the_nsa_att.html

IAB(*)によるインターネットの信頼性に関する宣言(2014/11)

- 新しくプロトコルを設計する際には、**暗号化機能を必須**とすべき。

- ネットワーク運用者やサービス提供者に**暗号化通信の導入を推進**するよう強く求める。

- コンテンツフィルターやIDS等平文通信が必要な機能については将来的に代替技術の開発に取り組む。

(* Internet Architecture Board)
https://www.iab.org/2014/11/14/iab-statement-on-internet-confidentiality/

なぜ全部HTTPSにしないといけないの？

- インターネットの健全性を確保するため。ここが揺らぐと大手のWebサービスの信頼性がなくなる。

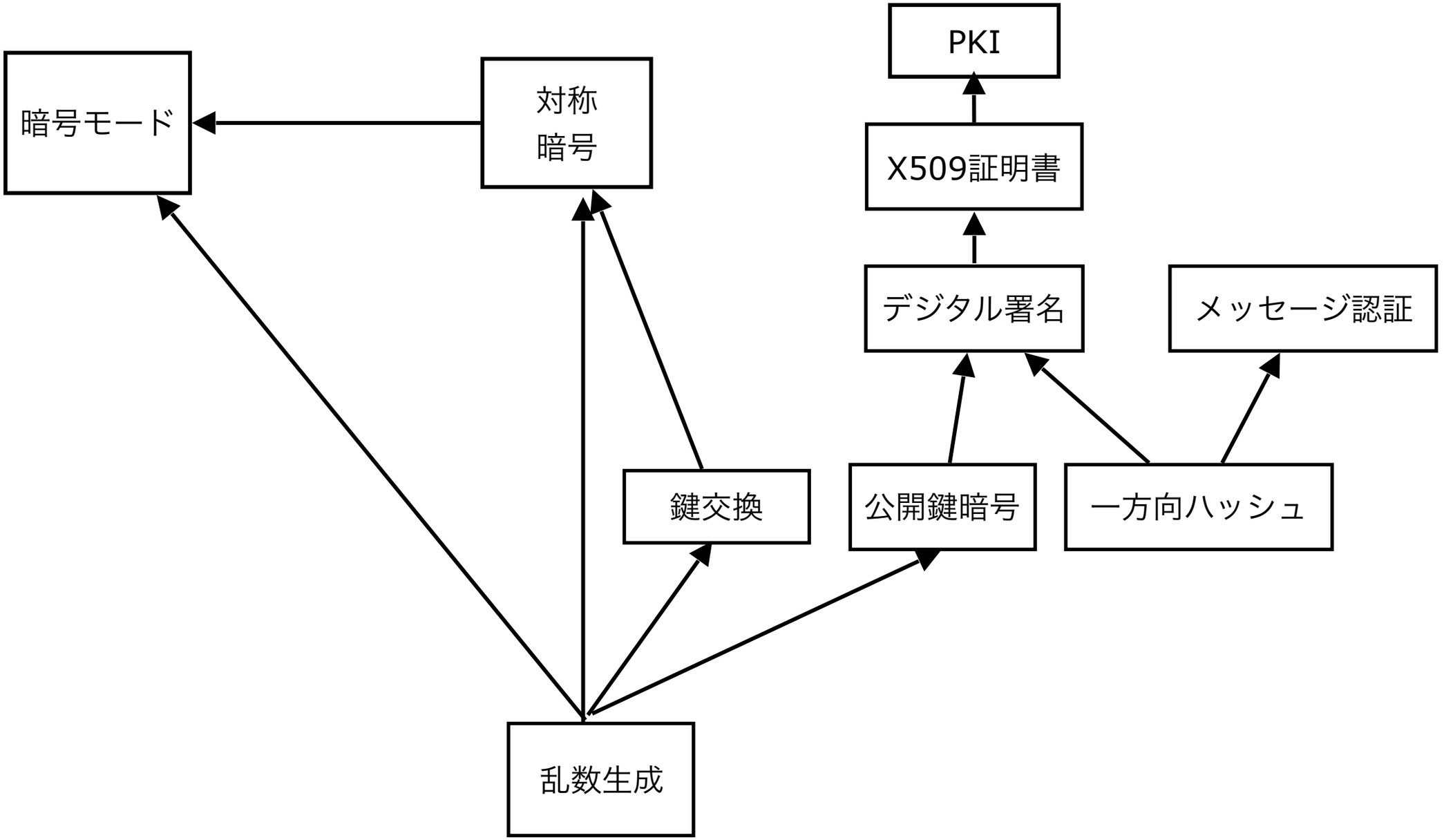
- 一部でも平文通信が残っているとそこを突かれる可能性がある。

- 自分は関係ないと思っていても知らないうちに仕込まれて利用される恐れも。

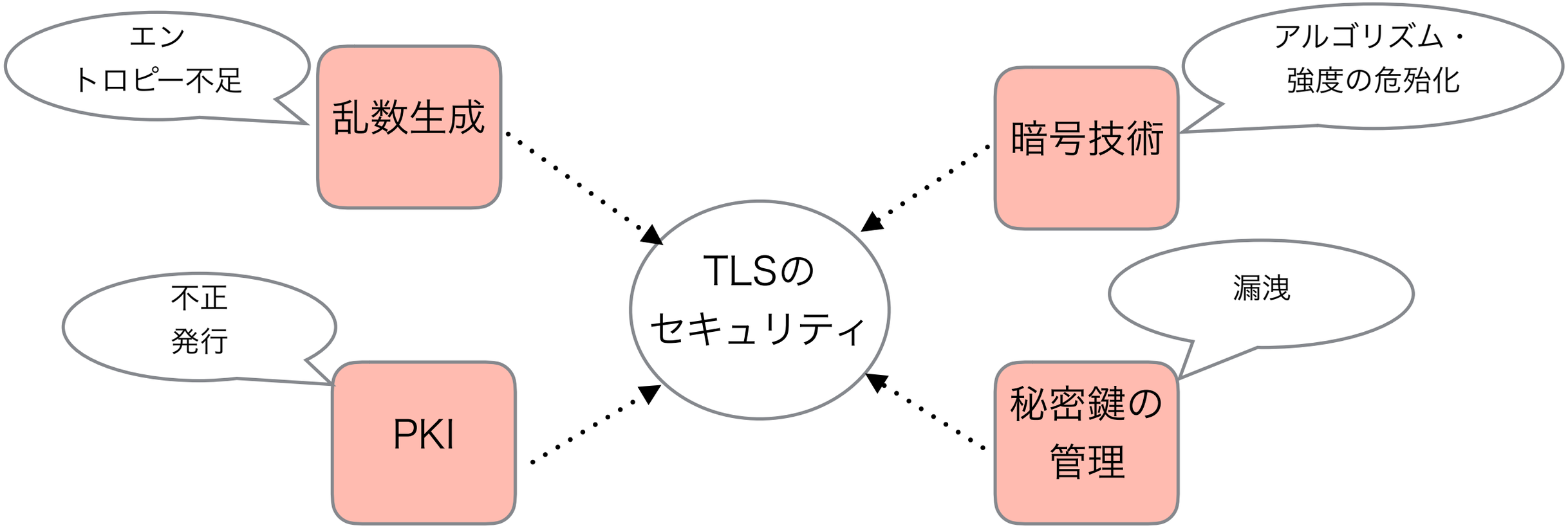
- 長期的な視点では http:// はフェーズアウトの方向です。

トラストアンカーをめぐる熱き戦い

TLSの技術要素


暗号モード
対称
暗号
PKI
X509証明書
デジタル署名
メッセージ認証
鍵交換
公開鍵暗号
一方向ハッシュ
乱数生成

TLSセキュリティの土台

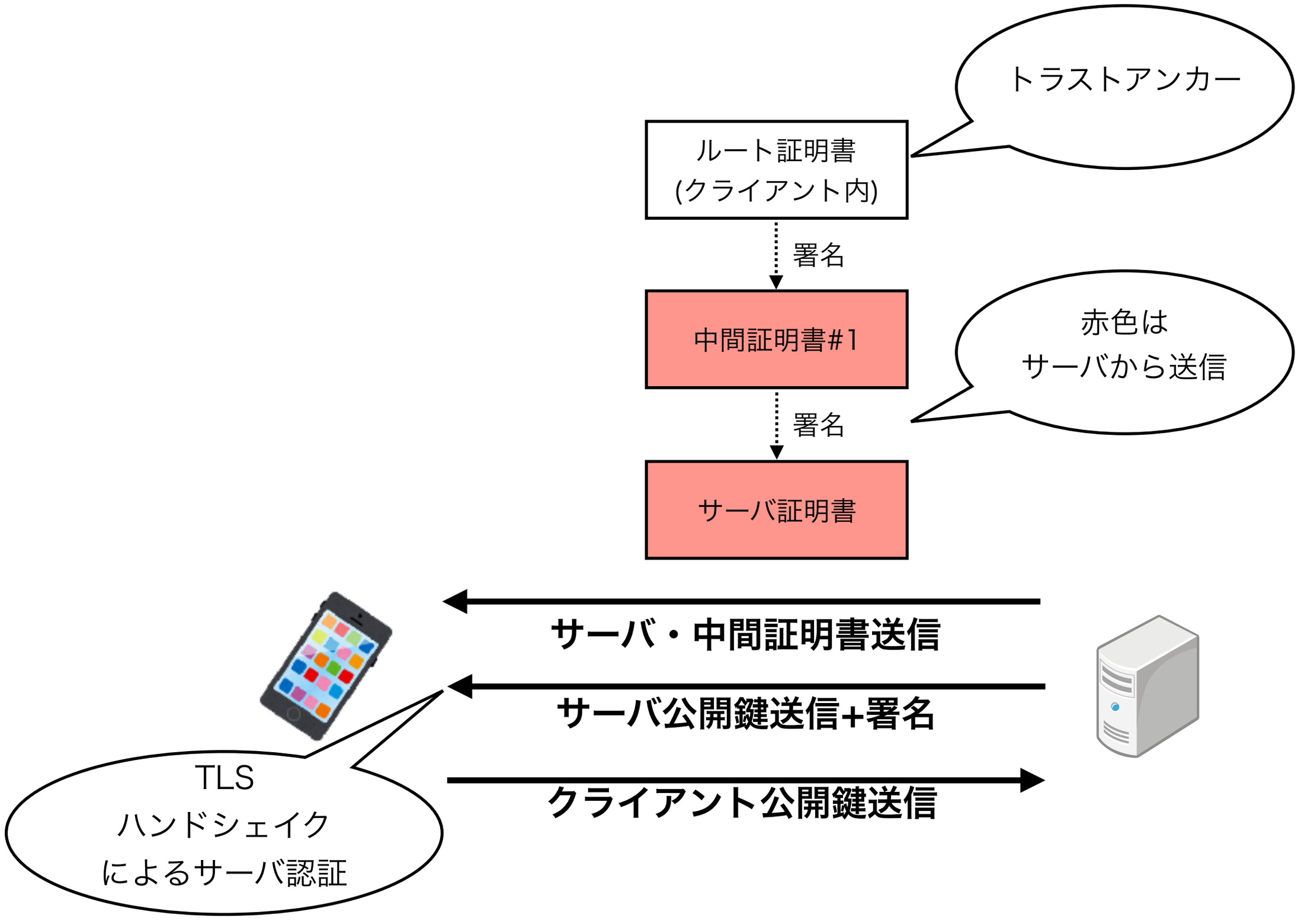


TLSは、この４つの外部要素の上でインターネットで安全な通信を提供する仕組みである。

逆に言えば、どれほど完璧なTLSプロトコルを作っても

この４つの外部要素が破られたら安全を確保できない。

信頼の要、トラストアンカー



TLSクライアントとOSで変わるrootCAの参照先

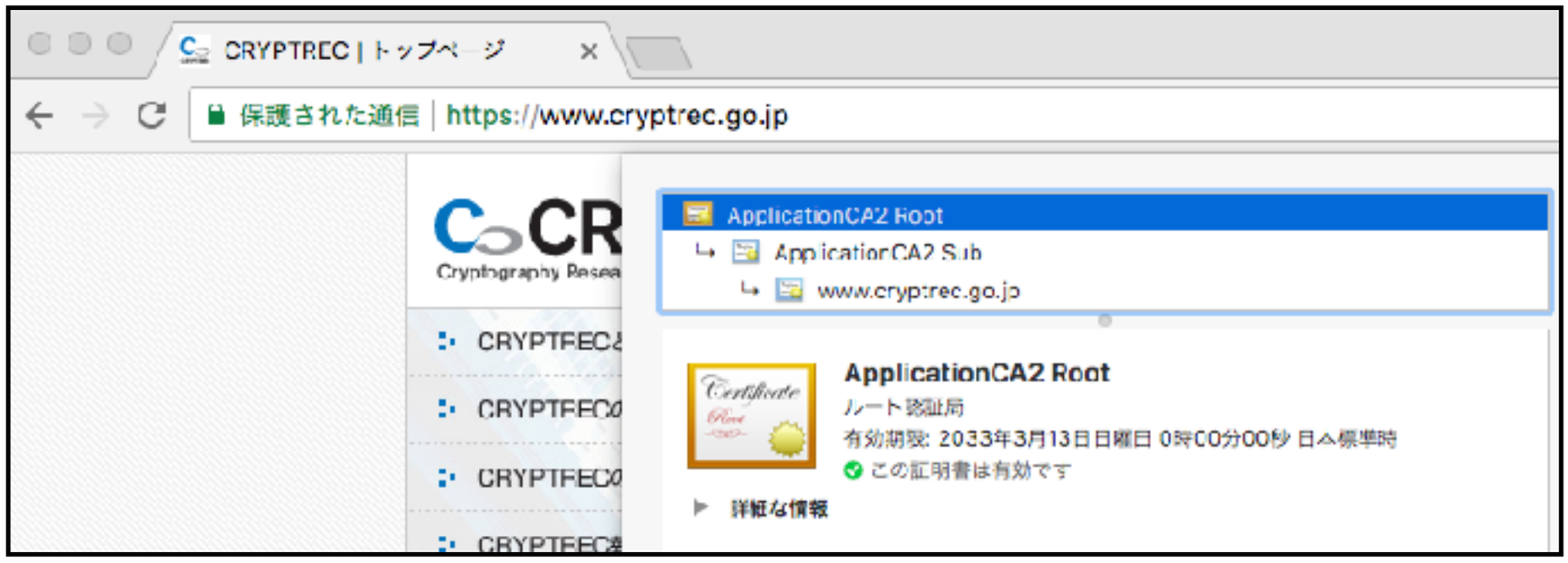
	Windows	MacOS	Linux	Android
IE/Edge	OSのrootCA	N/A	N/A	N/A
Safari	N/A	OSのrootCA	N/A	N/A
Chrome	OSのrootCA	OSのrootCA	Mozillaの rootCA(*)	Mozillaの rootCA(**)
Firefox	Mozillaの rootCA	Mozillaの rootCA	Mozillaの rootCA	Mozillaの rootCA
Node.js	Mozillaの rootCA	Mozillaの rootCA	Mozillaの rootCA	N/A

(* distribution提供のca-bundle pkgを参照する場合もあり), (** 変更、修正されている可能性あり)

MozillaのrootCAリスト https://hg.mozilla.org/mozilla-central/raw-file/tip/security/nss/lib/ckfw/builtins/certdata.txt

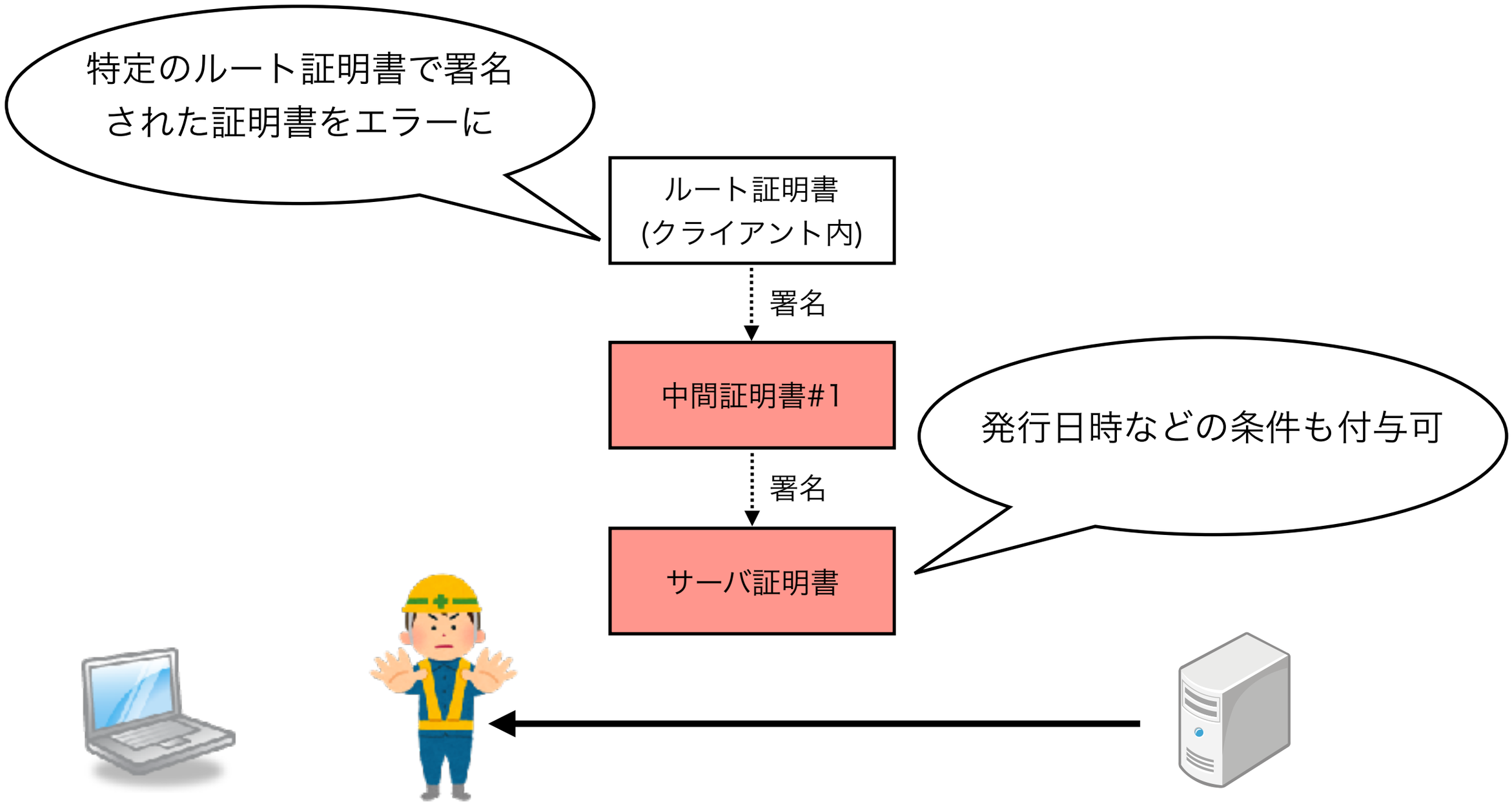
ブラウザによって異なる例

MacOS上のChrome



MacOS上のFirefox

ブラウザー vs 認証局

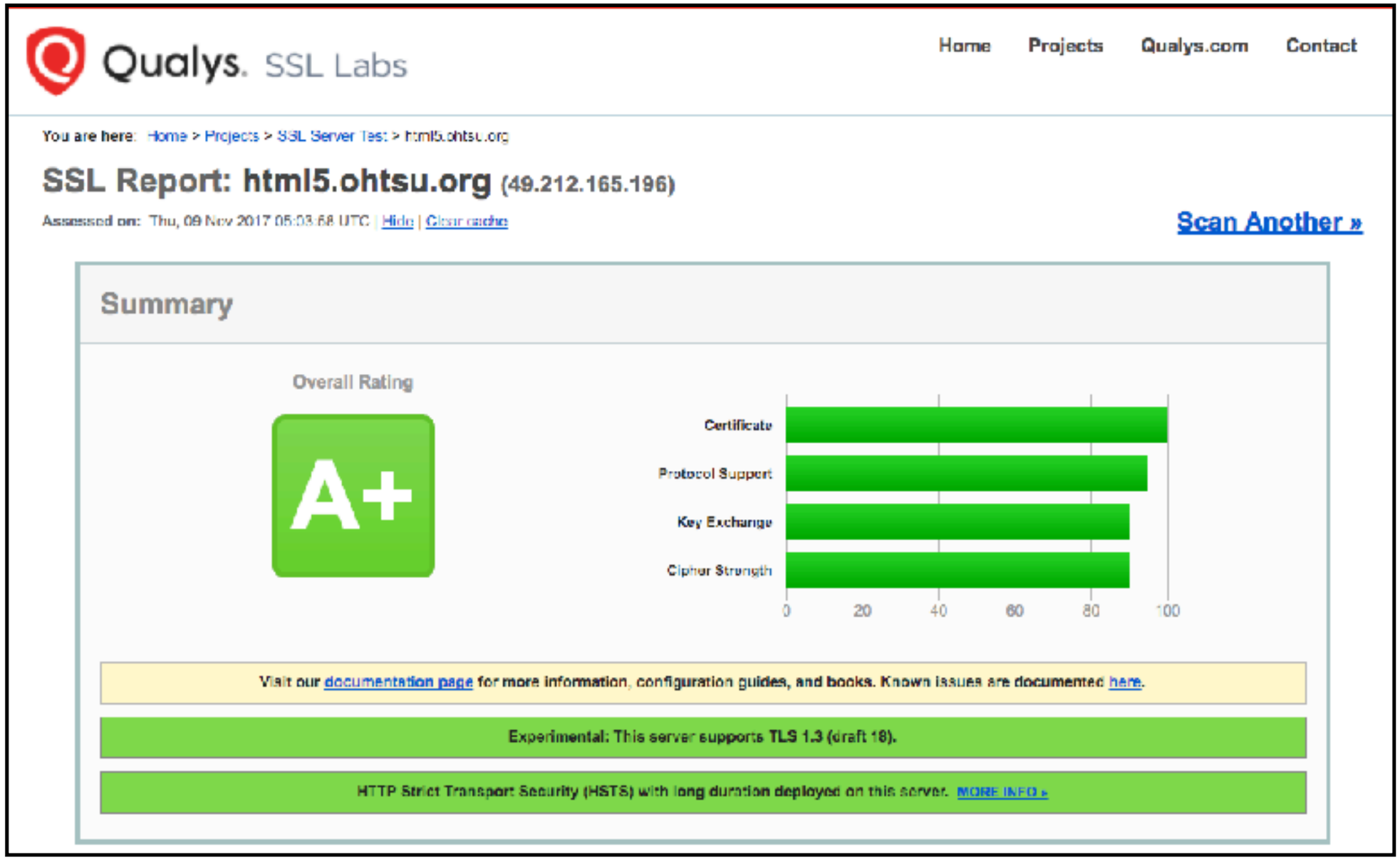


OSのroot証明書の管理とは別にクライアント独自の判断でフィルターできる。

具体的なrootCAのやらかし事件簿は、島岡さんのプレゼンで

HTTPSサーバの運用
技術負債
今後TLS1.0をどうしたらいいのか？

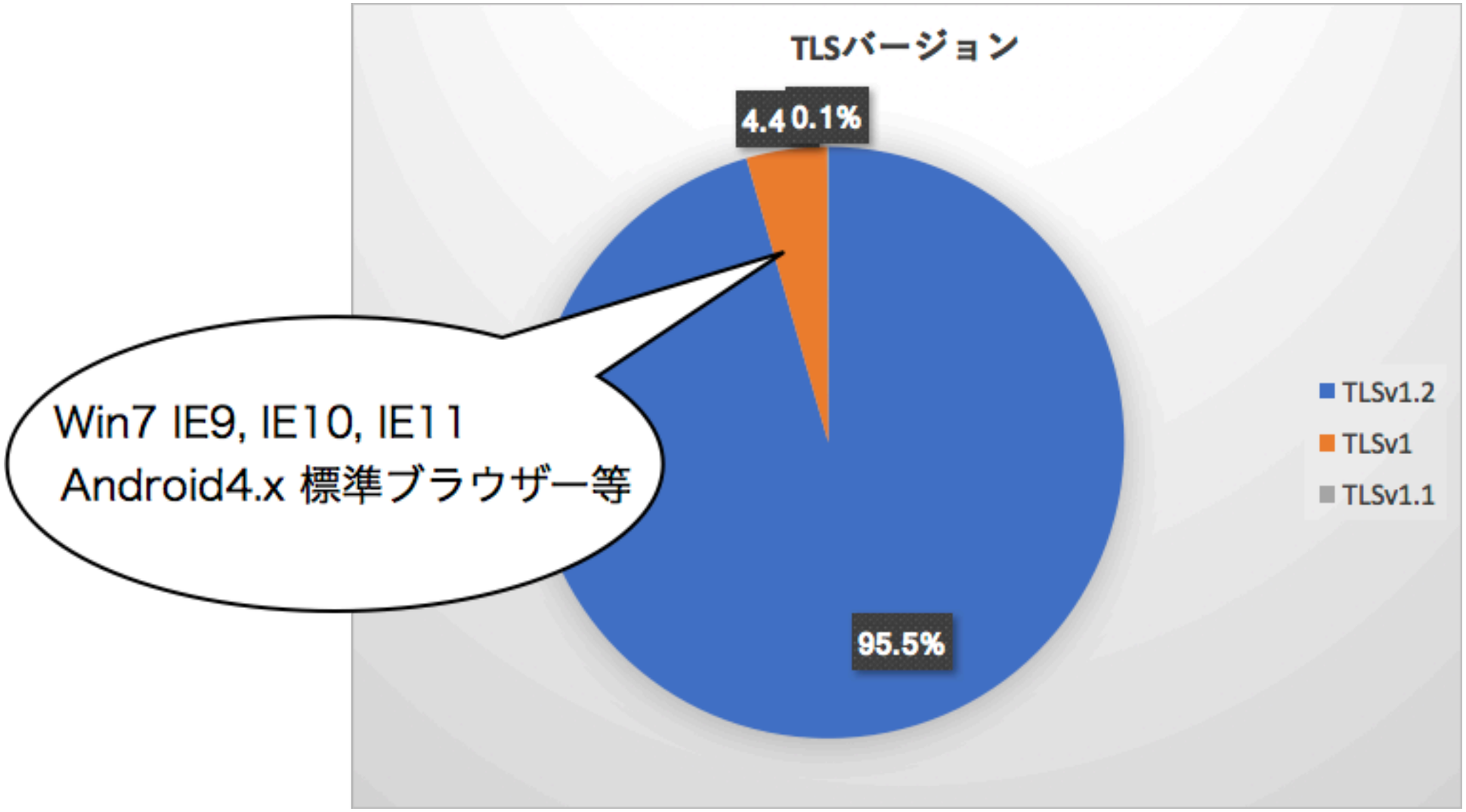
まずは



今の構成をしっかりチェックしましょう

技術負債の返済に備える

オワコンプロトコル(TLS1.0)



出典 https://www.slideshare.net/techblogyahoo/web-html5j-html5jb

TLS 1.0は、なぜオワコン？

- 1999年に仕様策定。もともとはSSLの標準化を目指したもの
- AES選定(2000年)前だったので当初はRC4とDES3のみサポート。RC4は既に危殆化、DES3はSWEET32攻撃を受けることが知られている
- ブロック暗号を使う際の初期ベクトルでBEAST攻撃を受けることが知られている(クライアント側で要対策)
- CBCモードの実装は過去数多くの脆弱性を生み出したのでAEAD(認証付暗号)への移行が主流に(*)
- MAC/PRF(鍵生成)ではMD5とSHA-1を組み合わせたもの。これがある限り危殆化されたMD5のコードをなくすことはできない(*)

(* TLS 1.1と共通項目)

TLSプロトコルの比較

	策定	ブロック暗号の 初期ベクトル	CBC モード	AEAD	MAC/PRF
TLS1.0	1999年	なし	有	なし	SHA1, MD5
TLS1.1	2006年	有	有	なし	SHA1, MD5
TLS1.2	2008年	有	有	有	SHA256以上
TLS1.3	仕様策定中	廃止	廃止	必須	HKDF, SHA256以上

PCI DSS3.1
禁止 2018/6〜

推奨

プロトコルの廃止は難しい

- もともとPCI DSS 3.1は2016年6月30日にTLS1.0を廃止予定だった。それを２年延期。

- 大手(SalesForce, IBM, Oracle)のサービスは既に対応済。だが移行作業はどこも一度は失敗してロールバックしている。

- Android4.xの標準ブラウザーが鬼門。API利用などはクライアント側の改修とか必要。

- 事前にユーザへの認知が難しい。廃止後はエラー画面に。

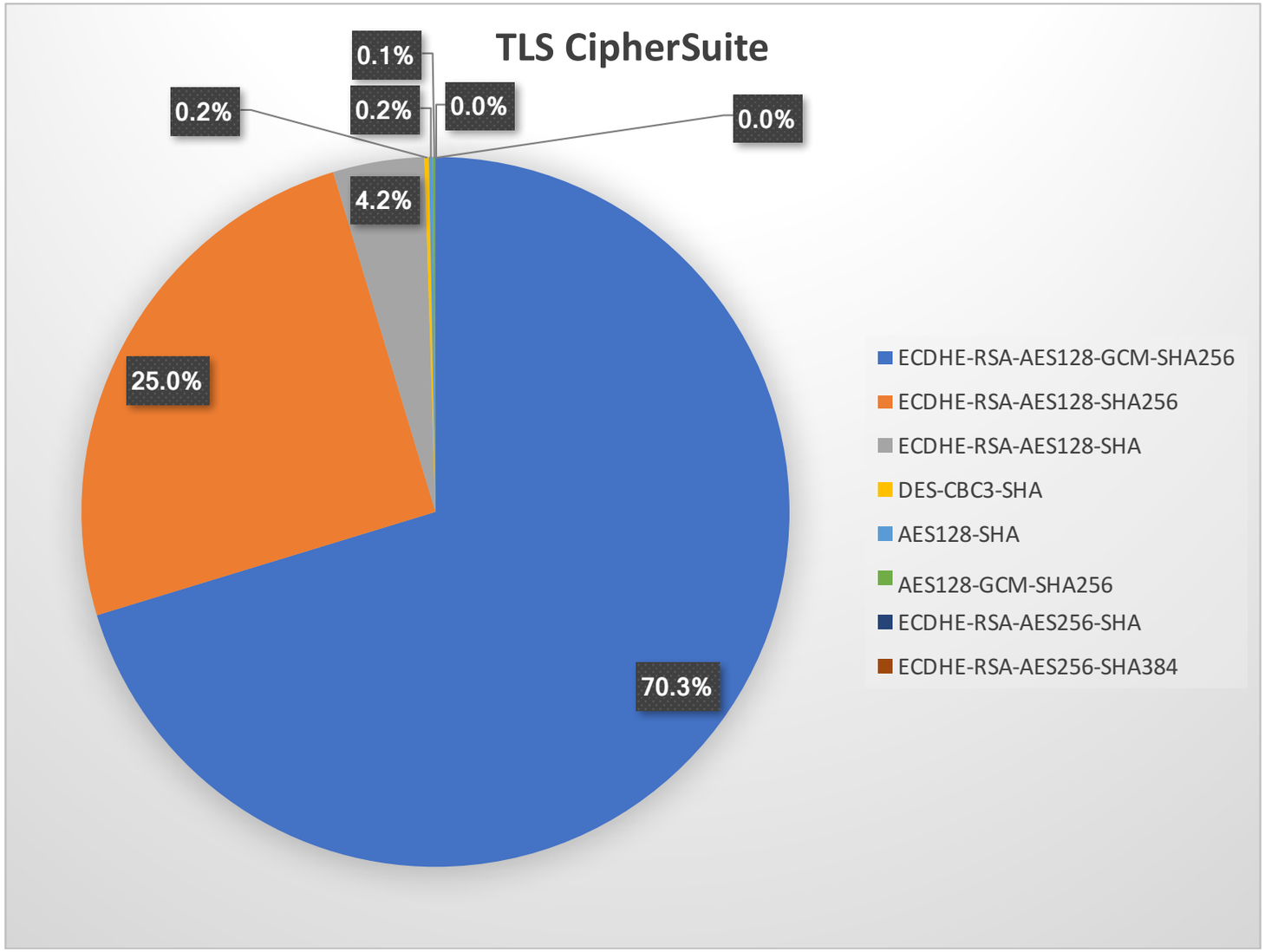
- SSL3.0はPOODLE脆弱性の公表でガラケー対応を含め廃止できた。TLS1.0は同じようになるのか、ならないのか。

ではどうすれば？

- まずはモニター
- TLSバージョンをアクセスログに出す
- 割合やUA、src ip等の傾向を把握
- インシデント発生時に影響度を把握できるよう準備が大事。

暗号方式のSPOF解消に向けて

暗号方式のSPOF



ECDHE-RSA-AES128-GCM-SHA256

鍵交換にECDHE
認証にRSA
対称暗号に128bit鍵長のAES
暗号モードにGCM(AEAD)
PRFハッシュにSHA256

出典 https://www.slideshare.net/techblogyahoo/web-html5j-html5jb

TLSの暗号方式の現状
SPOFの存在

鍵交換	RSA	Forward Secrecy			
		DHE	ECDHE NIST-P256	ECDHE x25519	
デジタル署名	RSA	DSS (DSA)	ECDSA	EdDSA (仕様化中)	
対称暗号	DES/RC4	DES3	AES	ChaCha20	その他
暗号モード	CBC	AEAD			
		CCM	GCM	Poly1305	
メッセージ認証 PRF	MD5	SHA-1	SHA256	SHA384	

赤: 危険, 黄: 注意, 緑: 安全, 青: これから

(注意は、暗号学的注意と将来的に普及が見込まれない注意も含まれます)

TLS暗号方式の現状(鍵交換)

鍵交換	RSA	Forward Secrecy		
		DHE	ECDHE NIST-P256	ECDHE x25519

RSA鍵交換: FSでないので非推奨。でも企業内、DC内で必要との声があり。
DHE鍵交換: 暗号強度の交換が不可。
ECDHE(NIST-P256): 一番普及している。NSAバックドアの疑義
ECDHE(x25519): やっと主流ブラウザでサポート開始。これから普及が見込まれる。

NIST-P256に何かあったらめちゃやばい

NIST標準としてx25519が採用される見込みが最近明らかに
https://csrc.nist.gov/News/2017/Transition-Plans-for-Key-Establishment-Schemes

TLS暗号方式の現状(署名)

デジタル署名	RSA	DSS (DSA)	ECDSA	EdDSA (仕様化中)	

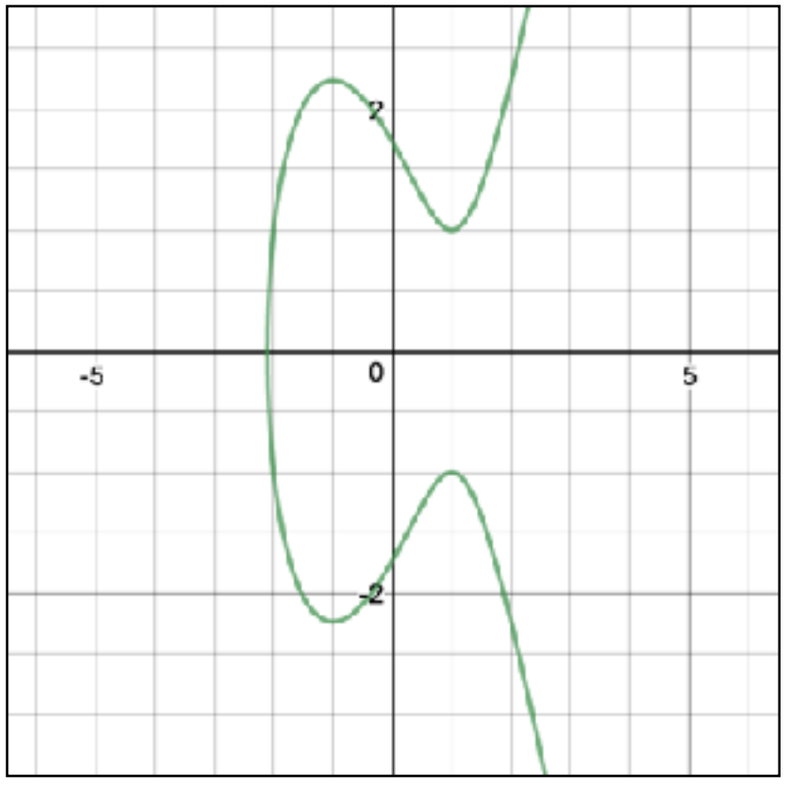
RSA署名: PKCS1v1.5方式が主流だが安全性が証明されていない。PSSへの移行が課題
DSA: ほとんど普及せず。性能的メリットもないため廃止の方向
ECDSA(NIST-P256): ECC証明書が将来もっと普及するか？
EdDSA(ed25519): 仕様化中(後述)

世の中の証明書がほとんどRSA署名
PKCS1v1.5になんかあったらめちゃやばい

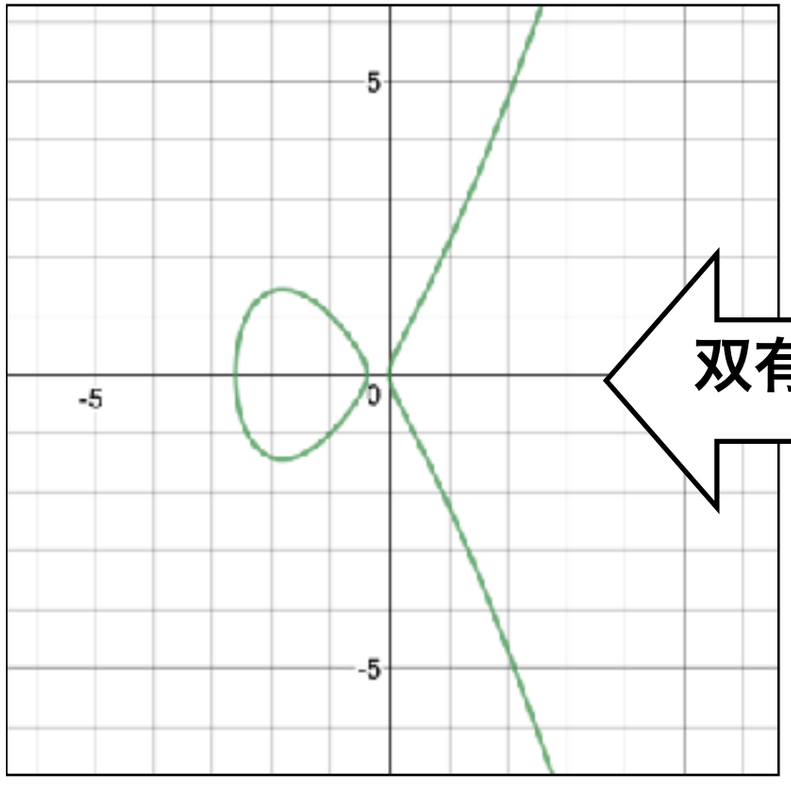
TLSで使う楕円曲線暗号の種類(一例)

時間の関係上スキップ

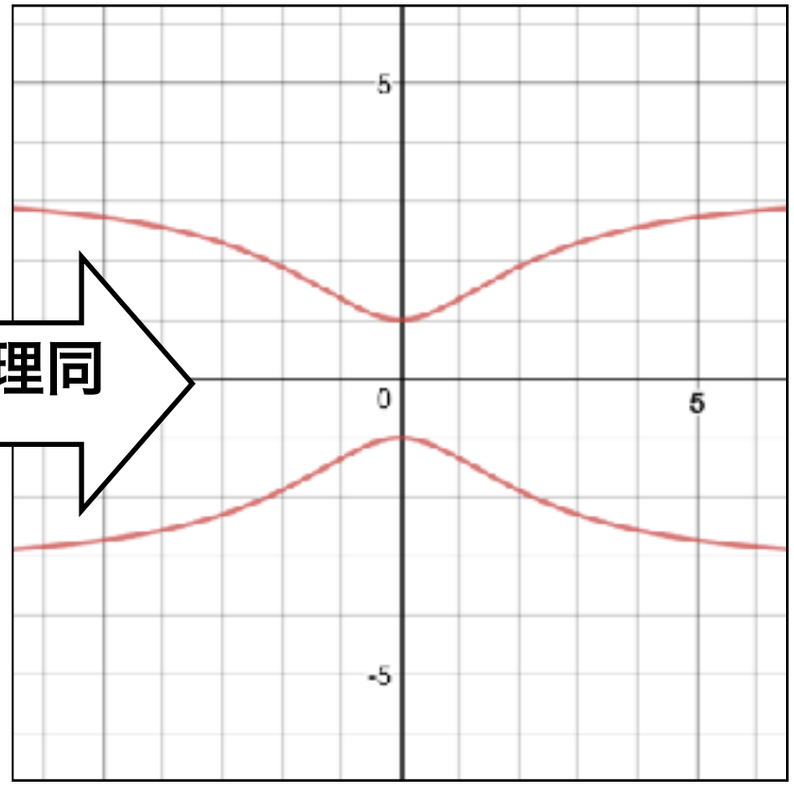
鍵交換・署名 (ECDHE・ECDSA)	鍵交換 (ECDHE)	署名 (EdDSA)
NIST P-256 (prime256v1, secp256r1)	X25519	Ed25519
FIPS 186-4	RFC7748	RFC8032
shortワイエルシュトラス曲線	モンゴメリ曲線	twistedエドワーズ曲線



$y^2=x^3+ax+b$



双有理同



$by^2=x^3+ax^2+x$

$ax^2+y^2=x^3+dx^2y^2$

(注：提示しているグラフは形が見やすいパラメータを使った楕円曲線を書いており、実際に使われる楕円曲線暗号のグラフと異なります。)

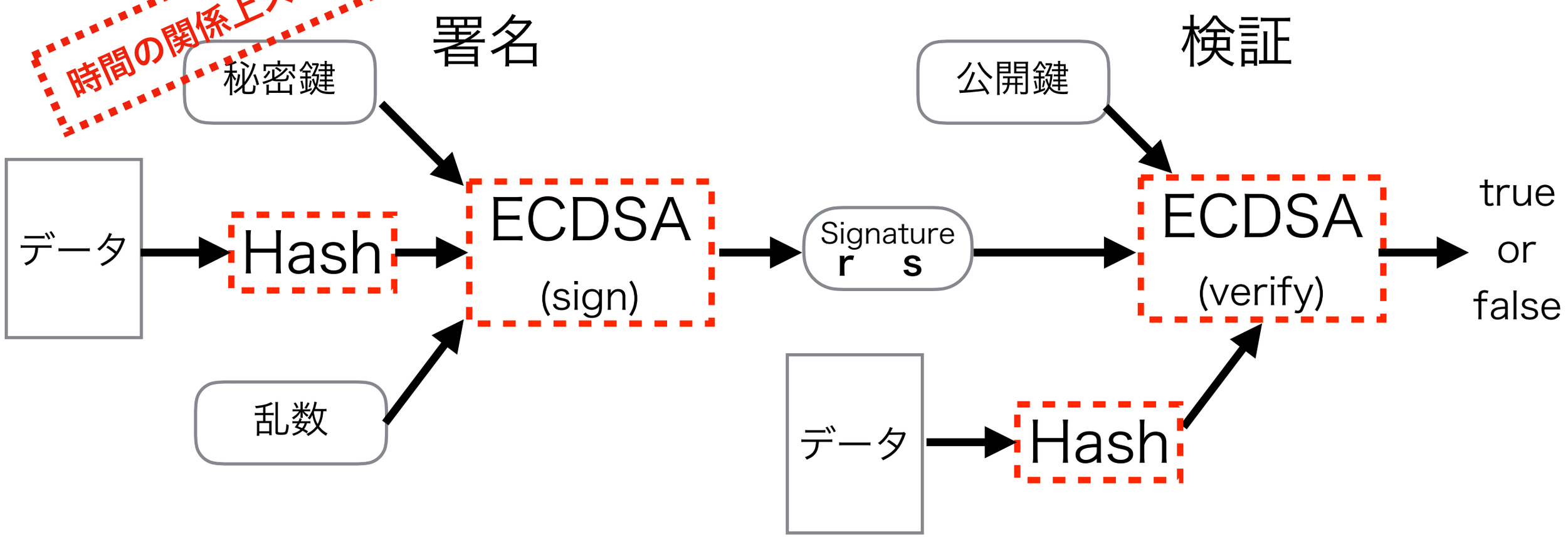
時間の関係上スキップ

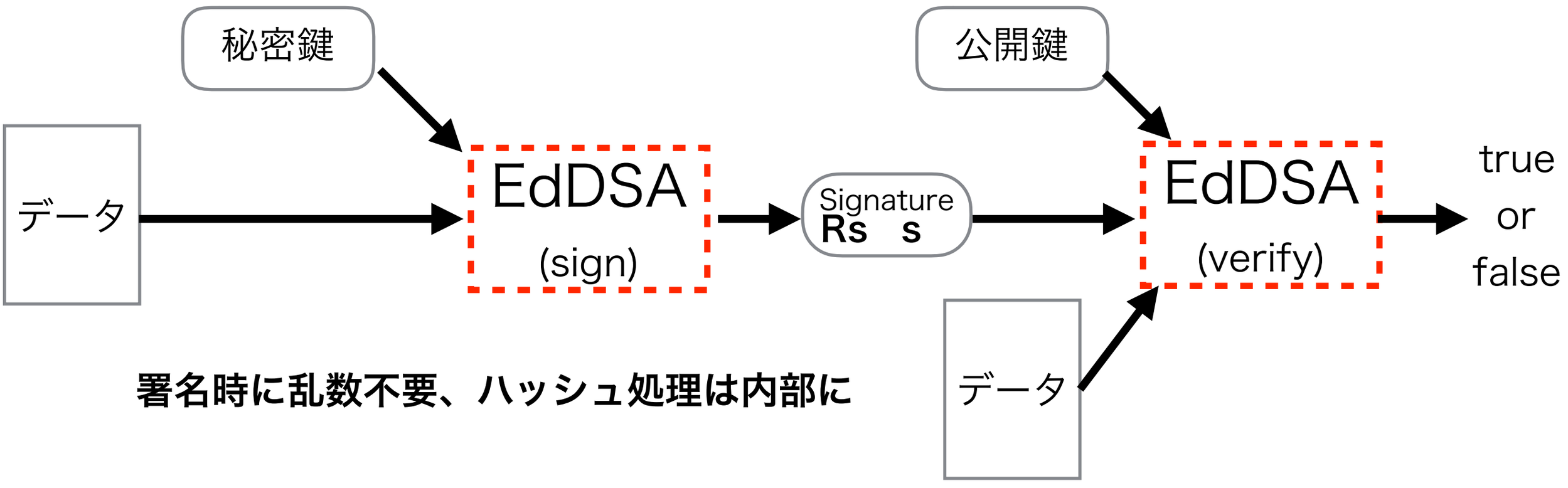
怪しい

	ECDSA NIST-P256	EdDSA Ed25519
セキュリティ強度	126bits	
楕円曲線・素数	$y^2=x^3-3x+41058363725152142129326129780047268409114441015993725554835256314039467401291$	$-x^2+y^2=1-(121665/121666)x^2y^2$
	$2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$	$2^{255}-19$
公開鍵書式・サイズ	uncompress 64bytes	compress(y座標) 32bytes
署名サイズ	64bytes	
特徴	署名毎に乱数が必要	同一秘密鍵・データを利用するとsignatureが同一
	ハッシュと組み合わせ	SHA-512のハッシュ処理を内部で持つ
	ハッシュ衝突耐性	ハッシュ衝突耐性+鍵偽造耐性
	要無限点処理	無限点処理が必要ない
	IUF(Input/Update/Finish)API	IUFでない
	高性能かつ安全な実装をするのが大変	サイドチャネルやキャッシュタイミング攻撃を比較的受けにくい
証明書	RFC3279、発行認証局あり	draft-curdle-pkix 仕様化中

ECDSA/EdDSA処理の違い

時間の関係上スキップ





署名時に乱数不要、ハッシュ処理は内部に

TLS暗号方式の現状
対称暗号・暗号モード

<table>
<tr><td>対称暗号</td><td>DES/RC4</td><td>DES3</td><td>AES</td><td>ChaCha20</td><td>その他</td></tr>
<tr><td rowspan="2">暗号モード</td><td rowspan="2">CBC</td><td colspan="3">AEAD</td><td rowspan="2"></td></tr>
<tr><td>CCM</td><td>GCM</td><td>Poly1305</td></tr>
</table>

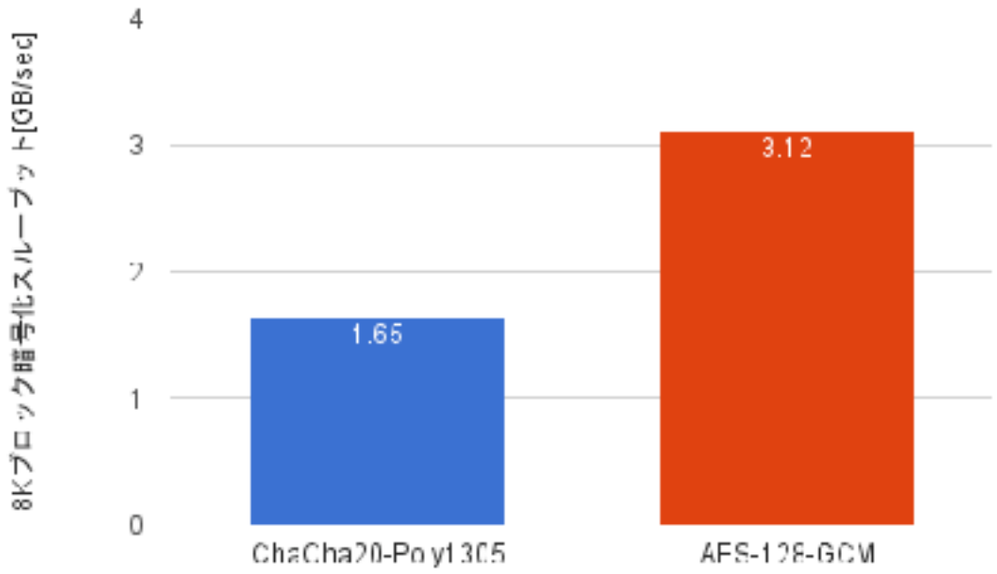
DES/RC4: 既に危殆化。利用禁止
DES3: Sweet32攻撃を受ける可能性があり。
AES-CBC: 過去CBCのパディング処理で実装の脆弱性があり
AES-CCM: 速度的なメリットがないため普及せず
AES-GCM: 現在一般的に使われている一択。HW処理で高性能。
ChaCha20-Poly1305： ブラウザーサポートが増え、これから普及の見込み。

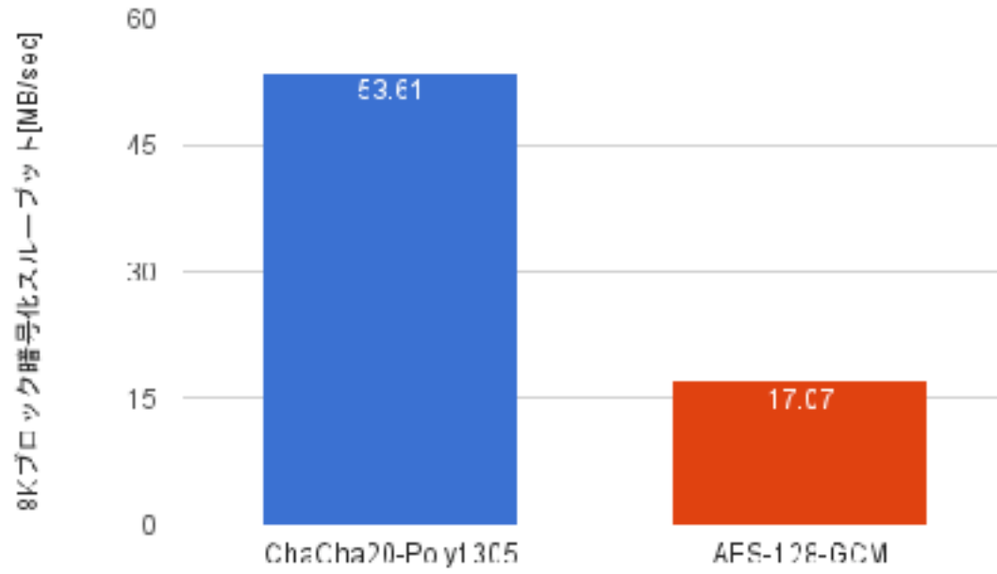
世の中ほとんどAES

AESに何かあったらメチャやばい

時間の関係上スキップ

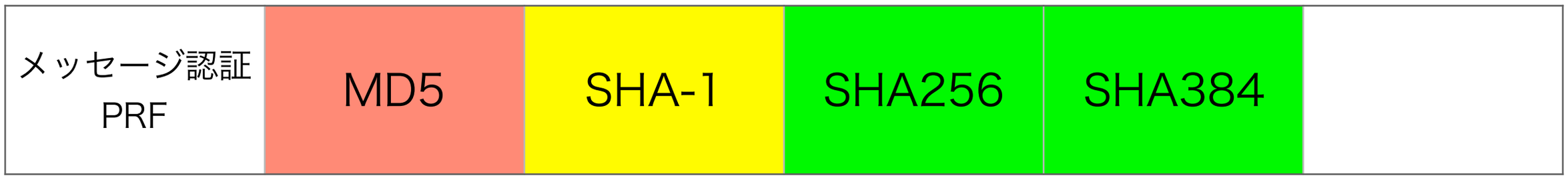
対称暗号性能比





- Intel AES-NIやARMv8 EncryptionエンジンなどHW暗号アクセラレーションを持つCPUではAES-GCMの速度が非常に速い
- HW暗号アクセラレーションがないCPU(ARMv7)などではChaCha20-Poly1305が速い

TLS暗号方式の現状
メッセージ認証・PRF



MD5: 既に容易に衝突計算が可能。利用禁止。
SHA-1: すったもんだの末、Scatterでトドメを刺され廃止に。
SHA-256,384: 一番普及している。当面安全と見られている。
SHA-3: 速度的なメリットがないためTLSへの導入機運は高まっていない。SHA-256系とアルゴリズムが異なり、伸長攻撃耐性があるのでバックアップアルゴリズムとしての期待はある。

SHA256が駄目ならSHA384があるさ

暗号方式のSPOF解消に向けて

- ChaCha20-Poly1350, x25519 のブラウザーサポートが充実してきた。
- OpenSSL-1.1.0ベースでは両方サポートされている。
- 2019年12月31日にOpenSSL-1.0.2がEOSLになる。
- wikileaksからスノーデン文書にNIST-P256のバックドアの詳細がもし出ようものなら大パニックになる。その前に x25519 への移行を。

将来に備える

TLS1.3

最新情報・どう判断すべきはパネルで

TLS1.3が求められる背景

1. 常時TLS時代を迎えるにあたって、しっかりしたプロトコルが必要

2. TLS1.2の限界

    - 様々な技術負債の蓄積

長期に使えるより安全で高性能なTLSプロトコルを作る

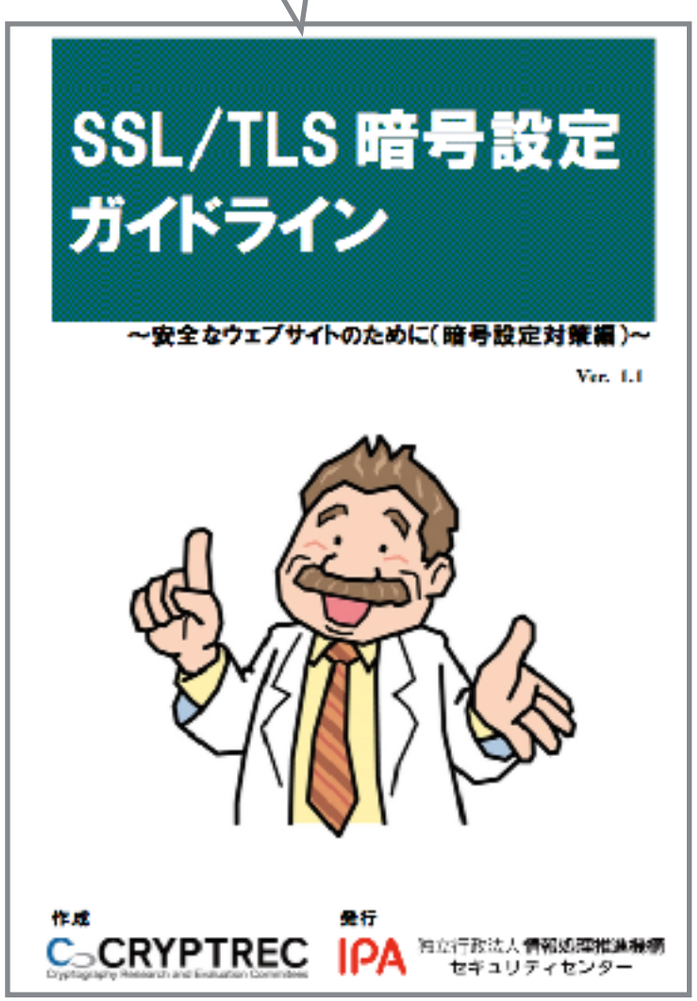
TLS1.2の限界

- 現在のTLS1.2で定義されている機能の一部は、既に利用すると危険である。

- 過去様々なTLSの攻撃手法や脆弱性が公開され、その都度対策が取られてきた。

- しかし一時的な対応で根本的・抜本的な対策になっていないものも多い。

本来このようなガイドラインがなくて済むのが望ましい



TLS1.3の特徴

1. 様々な機能、項目の見直し・廃止

   時代に合わなくなったもの、より効率的に変更修正できるものをTLS1.2から機能・項目を数多く廃止

2. よりセキュアに

   平文通信が必要な部分を極力少なくして情報を秘匿

   これまで攻撃対象となった機能を極力排除し将来的な攻撃に備える

3. 性能向上

   初期接続の短縮による性能向上

中身的にはTLS2.0レベルの大変更

TLS1.3に産業界(金融業界)からの懸念表明

- 「Industry Cancerns about TLS1.3」https://www.ietf.org/mail-archive/web/tls/current/msg21275.html

- Financial Service Roundtableの技術部門BITS担当者からTLS WGメーリングリストへの投稿

- RSA鍵交換がTLS1.3で廃止されると、金融業界で現状必要とされる要件に合わなくなる恐れがある。

- RSA鍵交換: サーバの秘密鍵を持っていればTLS通信データを復号化することが可能。TLS1.3では廃止予定。

- PFS(Perfect Forward Securecy): 一時的に有効な鍵を交換。後から通信データを復号化することは難しくなる。TLS1.3はECDHE/DHEのみ有効。

TLS1.3に産業界(金融業界)からの懸念表明(続き)

- 監視・モニター：金融業界では法的に従業員の通信データを保全する必要がある。

- トラブルシュート: アプリ層のエラーなどTLSデータの中身をデータセンター内で解析して障害調査を行っている。

- マルウェア、DDoS対策: TLSデータの中身を調べて検知してセキュリティ対策が行われている。

上記機能の運用は、現在RSA鍵交換を前提にシステムが構成されており、TLS1.3でRSA鍵交換が廃止されると大きな影響を受ける。議論中ですが、このまま行きそうです。

0-RTTのanti-replay

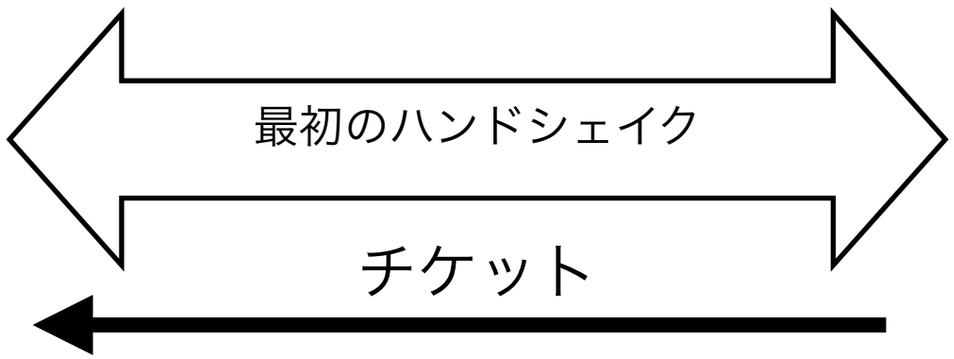
- TLS1.3ではハンドシェイク時にPSK(Pre Shared Key)を作成し、再接続時にハンドシェイクとデータを同時に送信する 0-RTT 機能が目玉として定義されている。

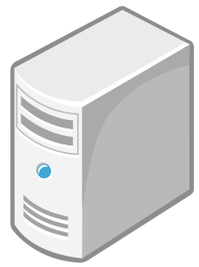
- 0-RTT機能によるデータ通信の高速化が期待されているが、0-RTTのデータを中間攻撃者が複製してサーバに送信するReplay攻撃に弱いという弱点が存在する。

- 0-RTTのreplay攻撃防御の手法としてはHTTPでは冪等性のあるリクエストに限定するといったようにアプリ側の対応が必要になる見込みである。

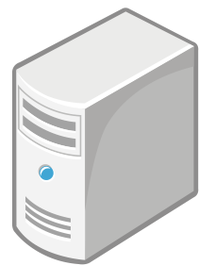
TLS1.3のハンドシェイク 0-RTT

時間の関係上スキップ





再接続



暗号化

ClientHello+チケット

0RTT

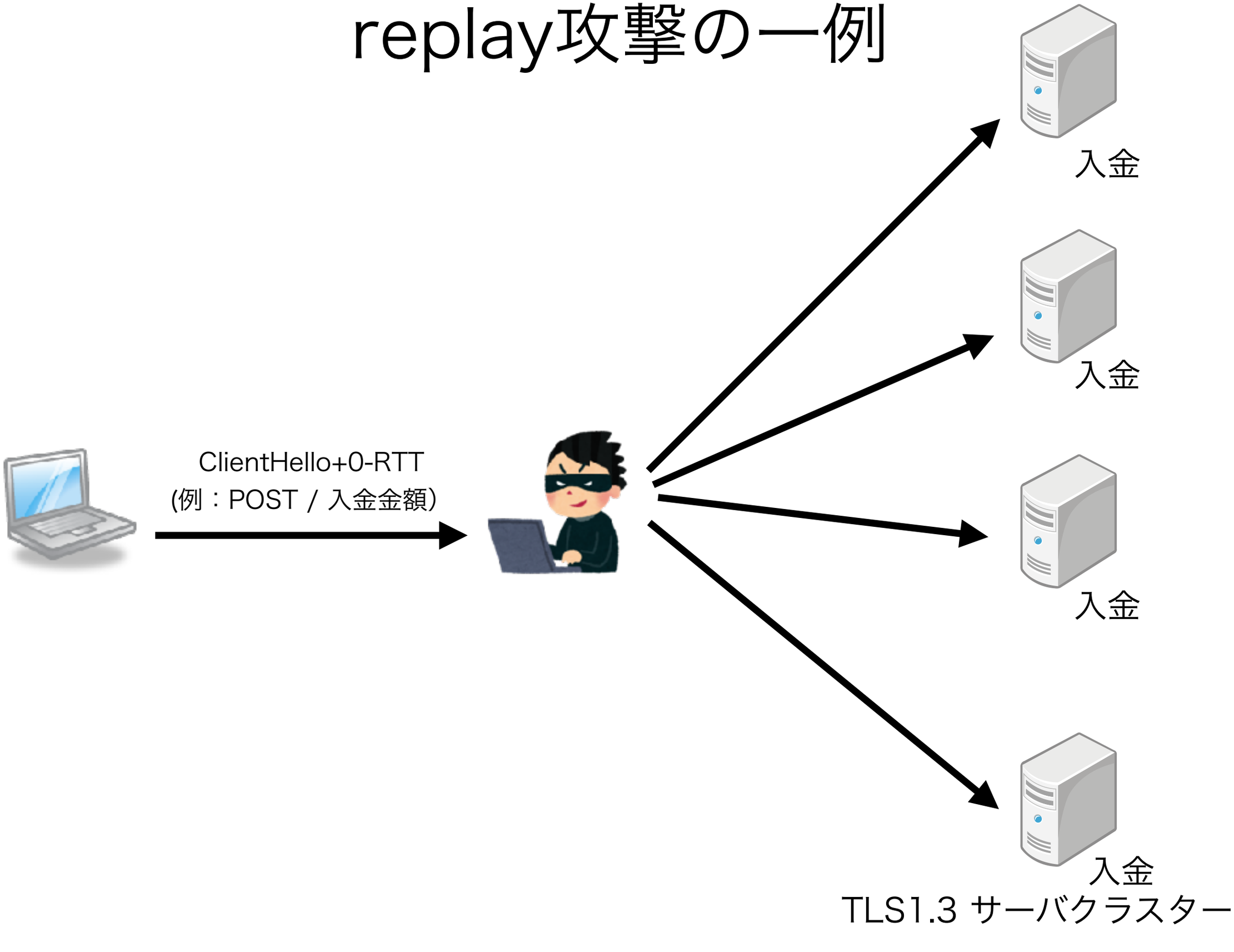
ApplicationData
end_of_early_data
Finished

0-RTT

ServerHello
EncryptedExtension
ServerConfiguration
Certificate
CertificateVerify
Finished
ApplicationData

1RTT

TLS 1.3 0-RTTの脅威
replay攻撃の一例



アプリ開発者やサーバ管理者が何を気をつければいいのか？

- 運用的な課題
  - TLS1.3が有効になることでミドルボックス、FW、IDS/IPSなど中間通信装置の障害や機能停止などの影響はないか？
  - TLS1.3で廃止になる機能(特にRSA鍵交換)を前提としたシステムの機能を使っていないか？
- 技術的な課題
  - 0-RTTを本当に安全に利用することができるのか？
  - 0-RTTを使わなかった場合に問題は発生しないのか？

移行のタイミングなど

- TLS1.3への移行タイミングについては、TLS1.3に何を求めるかによって異なる。
  - 将来的なセキュリティリスクの低減→TLS1.3
  - 0-RTTを使った高速通信の実現→TLS1.3
  - QUIC、WebRTCなど新しいプロトコルの導入→TLS1.3
- TLS1.3の仕様化によってTLS1.2が廃止されることはない。TLS1.2を継続して使い続けるのも一つの選択肢。TLS1.0/1.1はタイミングを見て廃止されるかもしれない。
- TLS1.2を使い続けることのデメリット: マーケット的にはTLS1.3を推進していくだろう。TLS1.2固有の問題が発見された場合にどうなるか？

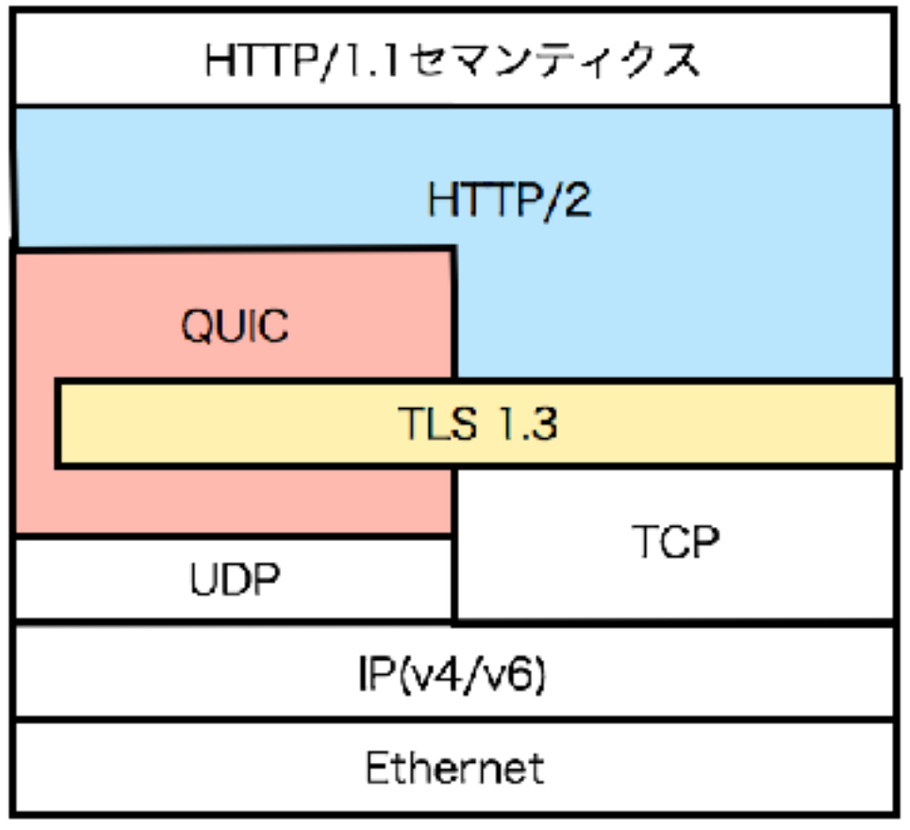
将来に備える

QUIC

最新情報・どう判断すべきはパネルで

QUICとは

- UDP上でTCP, TLS, HTTP/2の一部を実現するプロトコル。
- Googleが開発、2016年からIETF標準化が始まる。
- 現在Googleから出るトラフィックの30%以上がQUIC。これはインターネット全体のおよそ7%相当。



ユーザーランド実装 vs kernel実装

QUICのメリット
(パケットロスに強い)

回線輻輳

落ちたのを再送

パケット3

パケット5

パケット4

順番を気にせず待たされない

パケット2

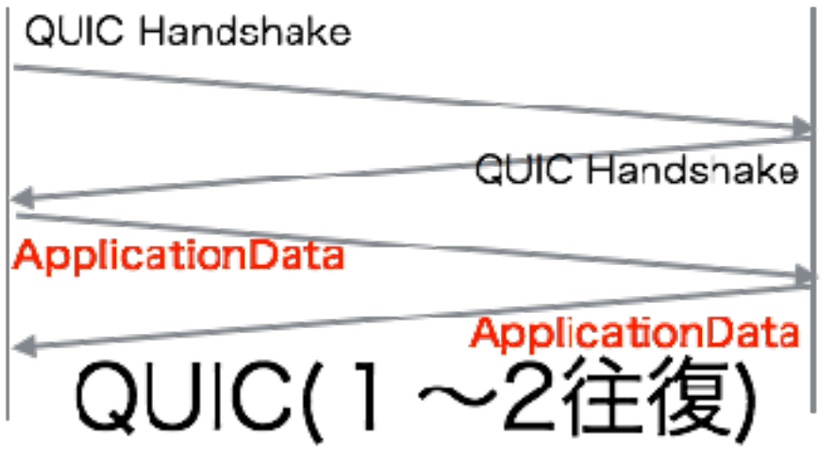
パケット1

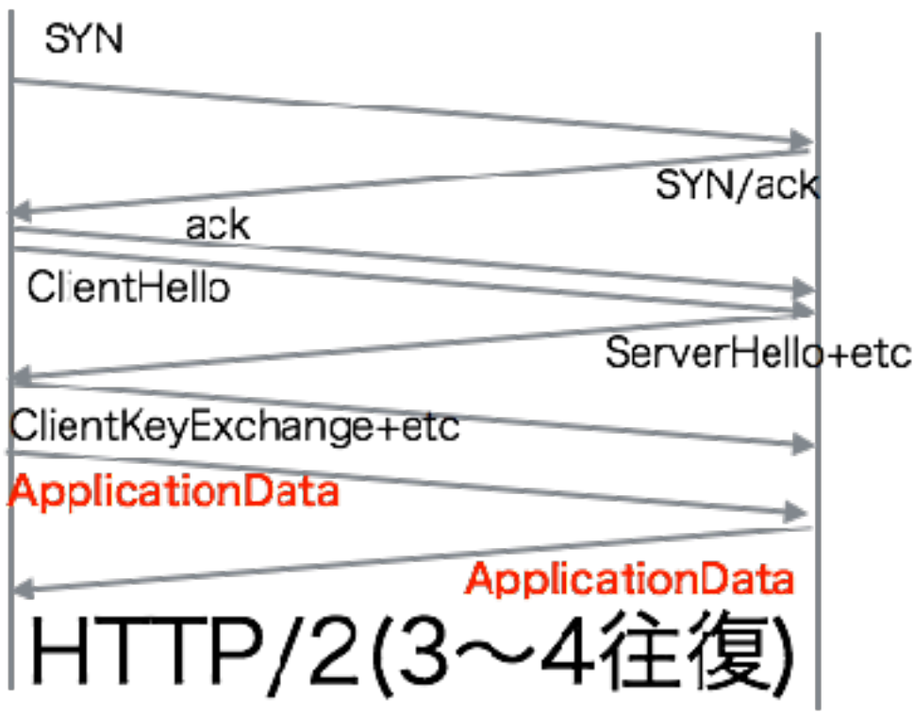
パケット3

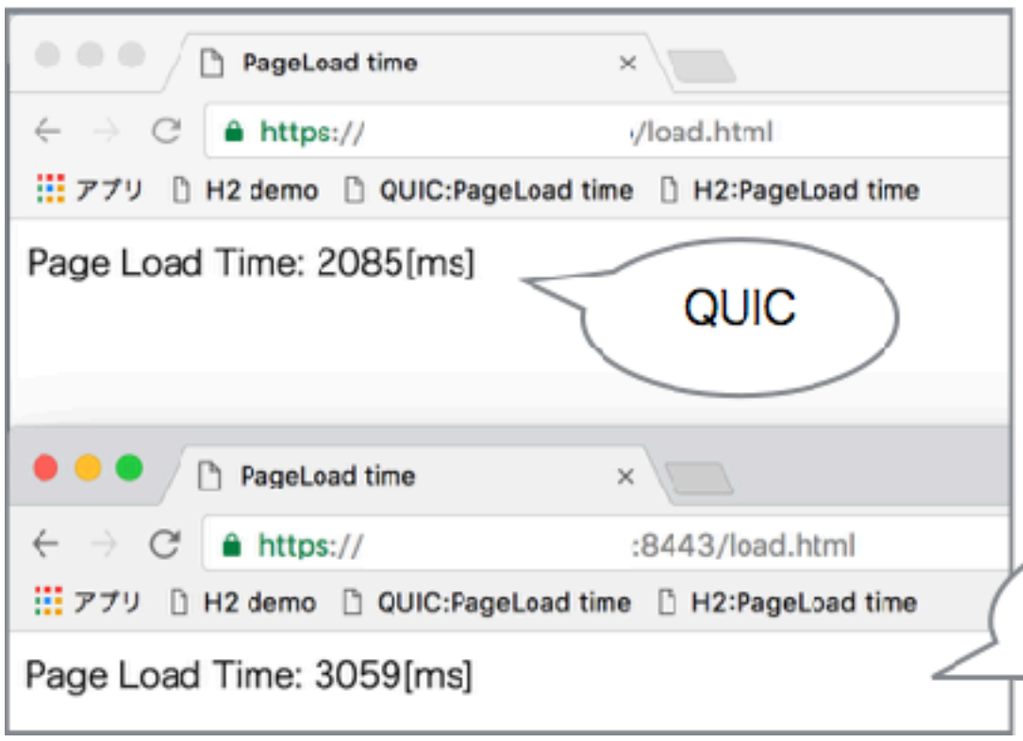
ドロップ

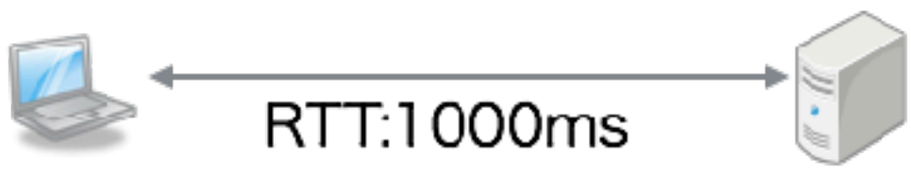
時間の関係上スキップ

QUICのメリット（高速接続）









さらに！ 0-RTTでもっと高速に

時間の関係上スキップ

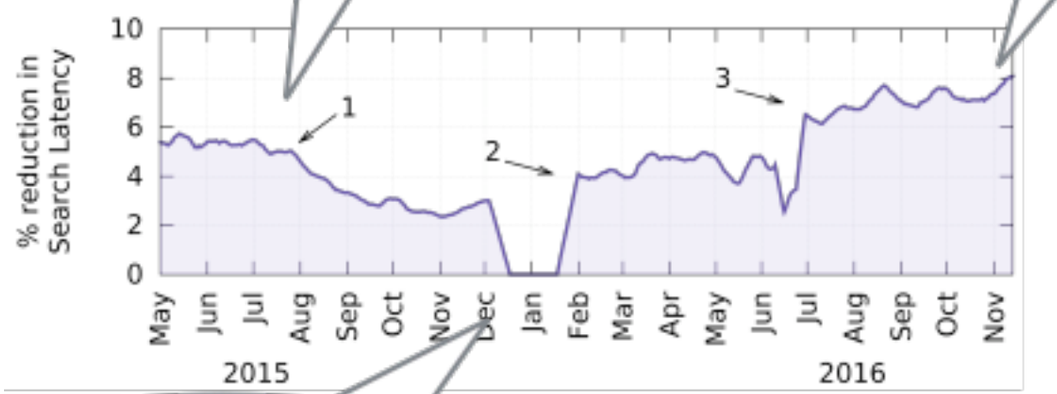
GoogleによるQUIC性能改善

- 検索レスポンスの改善割合(検索文字を入力してから結果が表示されるまでの時間)

設定ミスによるQUICの性能低下

平均8%改善

高レイテンシ接続での改善



脆弱性発見によるQUIC利用停止

		% latency reduction by percentile						
		Lower latency				Higher latency		
	Mean	1%	5%	10%	50%	90%	95%	99%
Search								
Desktop	8.0	0.4	1.3	1.4	1.5	5.8	10.3	16.7
Mobile	3.6	-0.6	-0.3	0.3	0.5	4.5	8.8	14.3
Video								
Desktop	8.0	1.2	3.1	3.3	4.6	8.4	9.0	10.6
Mobile	5.3	0.0	0.6	0.5	1.2	4.4	5.8	7.5

Googleは90%以上透過していると言うがQUIC切ったらChromeが軽くなったという声もチラホラ

出典：http://dl.acm.org/citation.cfm?id=3098842

将来に備える

耐量子暗号

耐量子暗号

- 量子コンピュータが Buzz word 化しそうです。
- まだまだ先の話との声もありますが、実用化されたら現在の暗号技術の脅威になります。
- 既に量子コンピュータ実用後に向けての準備が始まっています。
- いたずらに不安を煽り立てられないよう、今知っておくべきことを整理します。

ポスト量子 vs 耐量子

- post-quantum cryptography: 2003年に DJB が提唱し始める。PQCの略称も広まっている。

- quantum resistant crypto/quantum safe crypto等別名も提案されているが、まだ統一が図られていない。

- ある暗号アルゴリズムがその時点で量子コンピュータに対して安全とみなされても、その後もずっと安全である保証はない。量子コンピュータ実用後に必要なセキュリティレベルを指標としているので post-quantum だ(by djb)

量子コンピュータとは

- 量子ビット(qubit): $|\phi> = \alpha|0> + \beta|1>$ 0と1の重ね合わせた状態を持つ

- 量子コンピュータ: 量子ビットを使った演算を行う装置

1. 量子ゲート方式: 量子ビットを操作する基礎的な回路を組み合わせて汎用的な演算を行う量子回路を実現する方式。いくつかの**量子アルゴリズム**が提唱されている(後述)。IBM、Intel、MS等が取り組んでいる。

2. 量子アニーリング方式: 量子ビット間の相互作用を使って基底状態を探し出す方式。**組み合わせ最適化計算に特化**。カナダD-Wave社などが製品化。現在注目株。

(注: 上記以外の方式も研究開発中です。)

量子アルゴリズム

量子ゲート方式で考案されている量子アルゴリズムの一例

- Groverの探索アルゴリズム

量子ビット操作を繰り返して確率振幅を収束させ、N個のデータからN^(1/2)のオーダーでデータを探索できるアルゴリズム。対称暗号の共有鍵探索に応用可能。

- Schorの素因数分解アルゴリズム

  べき乗剰余計算を量子フーリエ変換を使って解き、素因数分解を行うアルゴリズム。有限体上の離散対数問題(DH)や楕円曲線暗号(ECDH)の解読にも応用可能。

量子コンピュータの暗号技術に対する脅威

暗号アルゴリズム	方式	用途	大型量子コンピュータの影響
AES	対称鍵暗号	暗号化	鍵長を大きくする必要がある
SHA-2, SHA-3	---	ハッシュ	ハッシュサイズを大きくする必要がある
RSA	公開鍵暗号	署名・鍵交換	安全でない
ECDSA, ECDH (楕円曲線暗号)	公開鍵暗号	署名・鍵交換	安全でない
DSA (離散対数問題をベースとした有限体暗号)	公開鍵暗号	署名・鍵交換	安全でない

出典 http://dx.doi.org/10.6028/NIST.IR.8105

量子コンピュータいつできる？

- 実用的な量子コンピュータの実現までだいたい20年ぐらいかかるだろうと言われている。

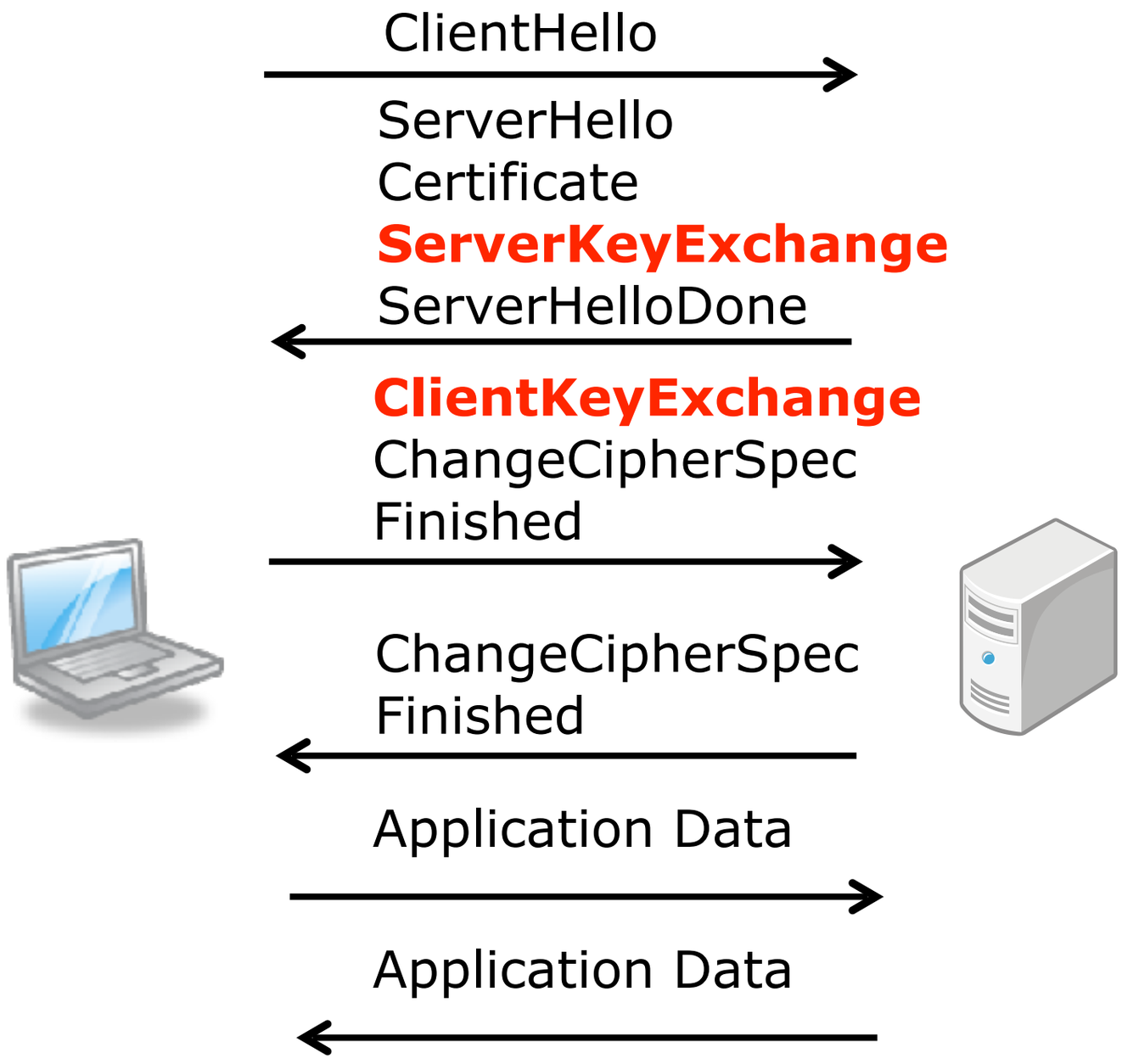
- IBMは、50 qubitのプロセッサーを完成(2017/11)。数多くの企業・機関が大量の資金を投じて開発中。

- DJBは、2033年までにRSA-2048が量子コンピュータで因数分解されるのに2048USDを賭けている(*)。

- 初期の量子コンピュータの普及で、新たな量子アルゴリズムや量子プログラミング言語の開発が進む可能性も。

(*) https://cr.yp.to/talks/2017.09.20/slides-djb-20170920-quantum-4x3.pdf

現在のTLSに対する脅威



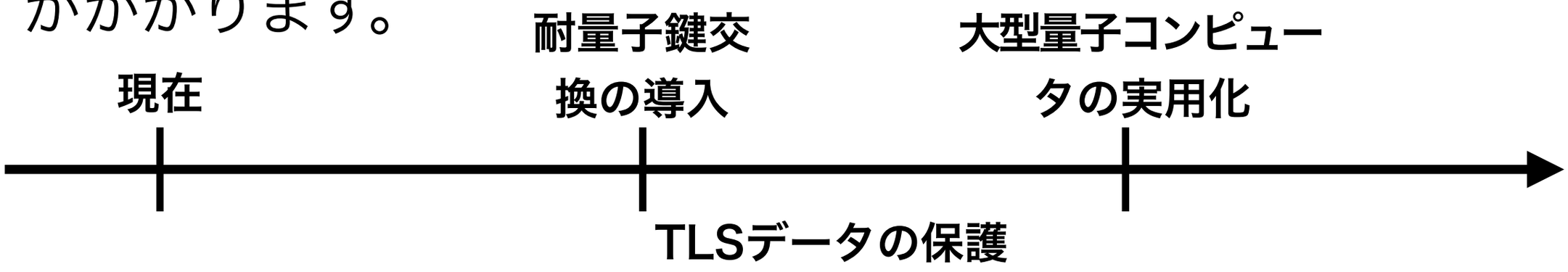
1. TLSハンドシェイクを含む全暗号データを保存
2. XX年後大型量子コンピュータ実現
3. ServerKeyExchange中の一時的公開鍵データからSchorのアルゴリズムを使って量子コンピュータから一時的秘密鍵情報を計算
4. ClientKeyExchange中の一時的公開鍵と組み合わせてpre_master_secretを計算
5. 暗号化されたApplication Dataで使われている対称暗号の共通鍵を計算。Application Dataを復号して平文情報を取得。

量子コンピュータの実現前に対応を

- 現在TLSでやり取りされる暗号データは、ハンドシェイクまで保管されているとXX年後には量子コンピュータで平文がわかってしまいます。

- 耐量子暗号な鍵交換アルゴリズムは、量子コンピュータの実現前に導入しておく必要がある。

- 新しい暗号方式が普及するまでかなり時間（数年単位）がかかります。



時間の関係上スキップ

GoogleによるCECPQ1試験

- CECPQ1: Combined Elliptic Curve and Post-Quantum
- 楕円曲線暗号のx25519と耐量子暗号RLWE(Ring Learning With Error)のNew Hopeを組み合わせてChromeのTLS鍵交換を試験
- x25519は32バイト、New Hopeは128バイトの公開鍵。合計160バイト長の鍵交換は通常より大きいため影響を図るのが目的
- NISTの選考(後述)が開始されるし、Chromeの試験がデファクトなってしまうのを避けるため終了。2016年5月〜2016年12月
- 接続レイテンシーが数ミリ秒遅くなった。遅い接続ほどその影響がお大きいことが判明。

https://security.googleblog.com/2016/07/experimenting-with-post-quantum.html
https://www.imperialviolet.org/2016/11/28/cecpq1.html

NISTによるPost-Quantum Cryptography Standardization

Competition (一つを決めるわけ)ではない。複数アルゴリズムの優劣など評価することになる。

- 2016/12/20 応募開始
- 2017/11/30 応募締め切り
- 3〜5年 分析フェーズ
- 2年後 Draft Standardが完成

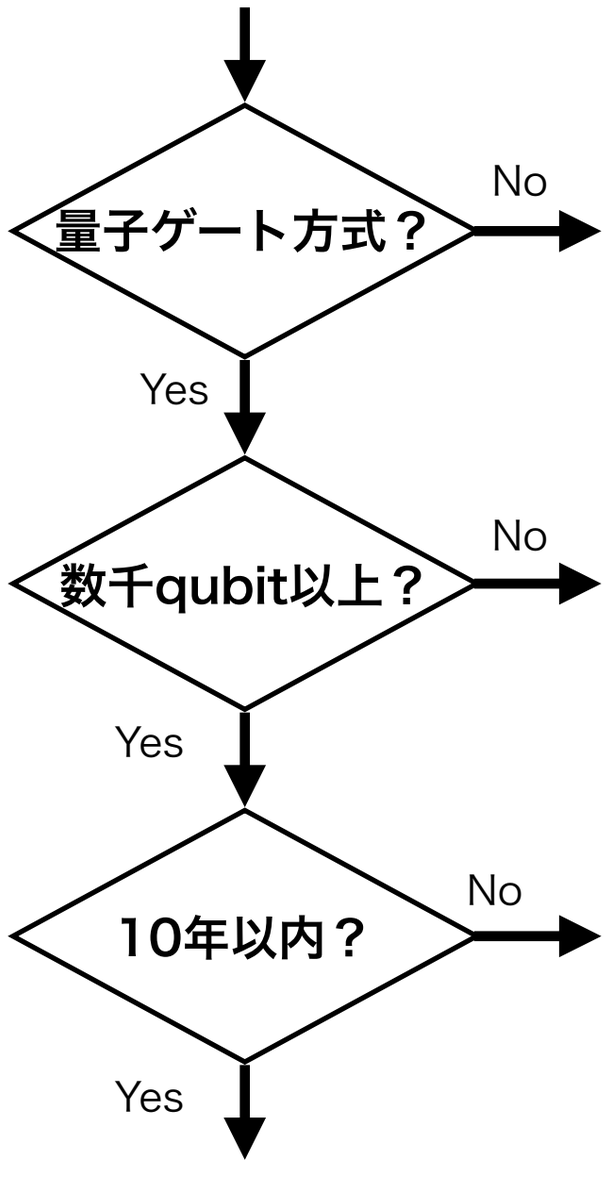
https://csrc.nist.gov/projects/post-quantum-cryptography

耐量子暗号候補

暗号方式	利用している技術
ハッシュベース署名	ワンタイムハッシュのみ利用
格子暗号	最短ベクトル探査問題
コードベース暗号	誤り訂正符号理論
多変数暗号	有限体上の多変数連立2次方程式
同種写像暗号	超特異楕円曲線

意識高い系上司からの問いに備える

量子コンピュータができるらしいぞ。TLS通信はどうなるんだ？



あぁ、あきらめましょう orz

知らないと困る？！ 認証局とHTTPSの最新技術動向

パネルディスカッション
～Webサーバ運用の観点で～

大津 繁樹
ヤフー株式会社

Internet Week 2017
2017年11月29日

IETF100後の
TLS1.3の最新動向(2017/11/28)

- 技術仕様は、ほぼ確定
- 途中の middle box がTLS1.3を切断する透過性の問題が明らかに。
- Google/Mozillaが各種パターンを試して透過性を数ヶ月測定
- TLS1.2にそっくりにするよう(意味のない)データを付与させると透過率が向上する見込みであることがわかる。
- 仕様を更新、再度測定し３回目の Last Call に。これで本当に確定する見込み。

サーバプログラム担当者に TLS1.3を使いたいと言われたら

- TLS1.3導入ポイントのスライド (アプリ開発者やサーバ管理者が何を気をつければいいのか？、移行のタイミングなど）をチェック

- TLS1.3の仕様化後は各種ブラウザですぐサポート予定。OpenSSL-1.1.1のリリースもTLS1.3完了待ち。

- NISTガイドライン案 https://csrc.nist.gov/publications/detail/sp/800-52/rev-2/draft

  - 2020年1月1日までにTLS1.3への移行を求める。

今後急速にTLS1.3の普及が見込まれるので覚悟を

IETF100後の
IETF QUICの最新動向(2017/11/28)

- 2018年3月の基本仕様の策定完了予定を2018年12月に延期する。

- 今後のIETF QUICのバージョンアップで変わらない部分を確定させる。

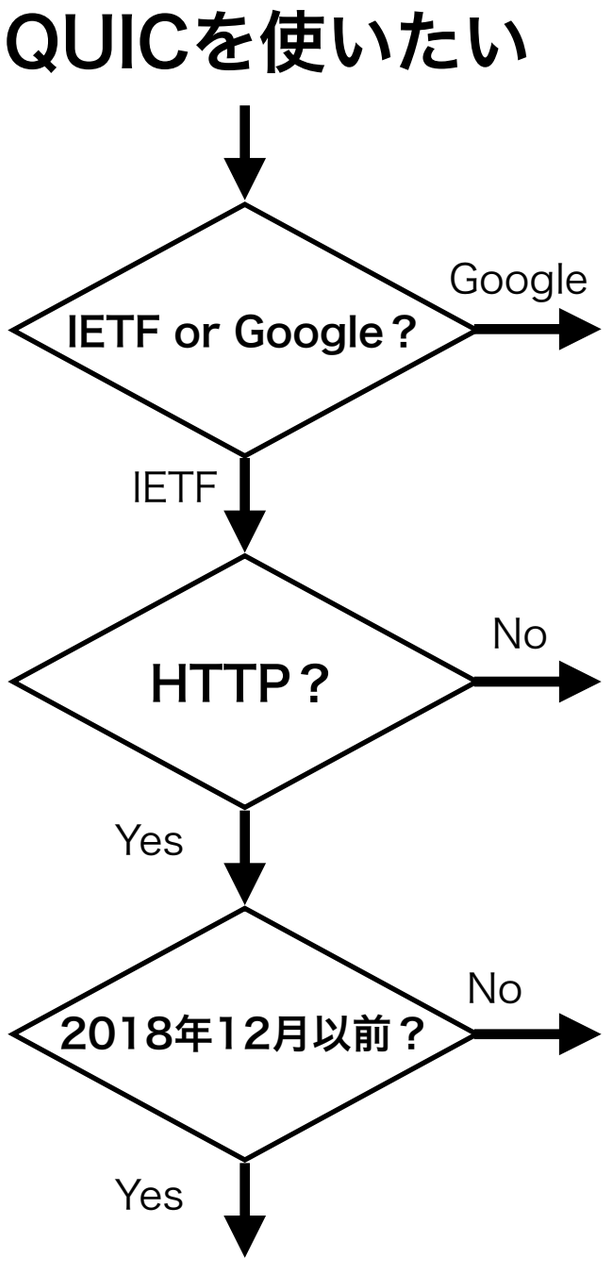
- IETF QUIC v1 は HTTP 対応に限定。ただし将来的にHTTP以外も使えることも配慮する。

- プロトタイピングや中間会議でもっと仕様化作業をもっとすすめる。

サーバプログラム担当者にQUICを使いたいと言われたら?

- IETF QUIC と Google QUIC の違いわかってる？
- Google QUIC は茨の道
- IETF QUICは今まさに仕様化作業中

サーバプログラム担当者からの要望備える



Chromeの開発が活発でQUICのバージョンアップが
次々行われるので追随するのが大変よ。

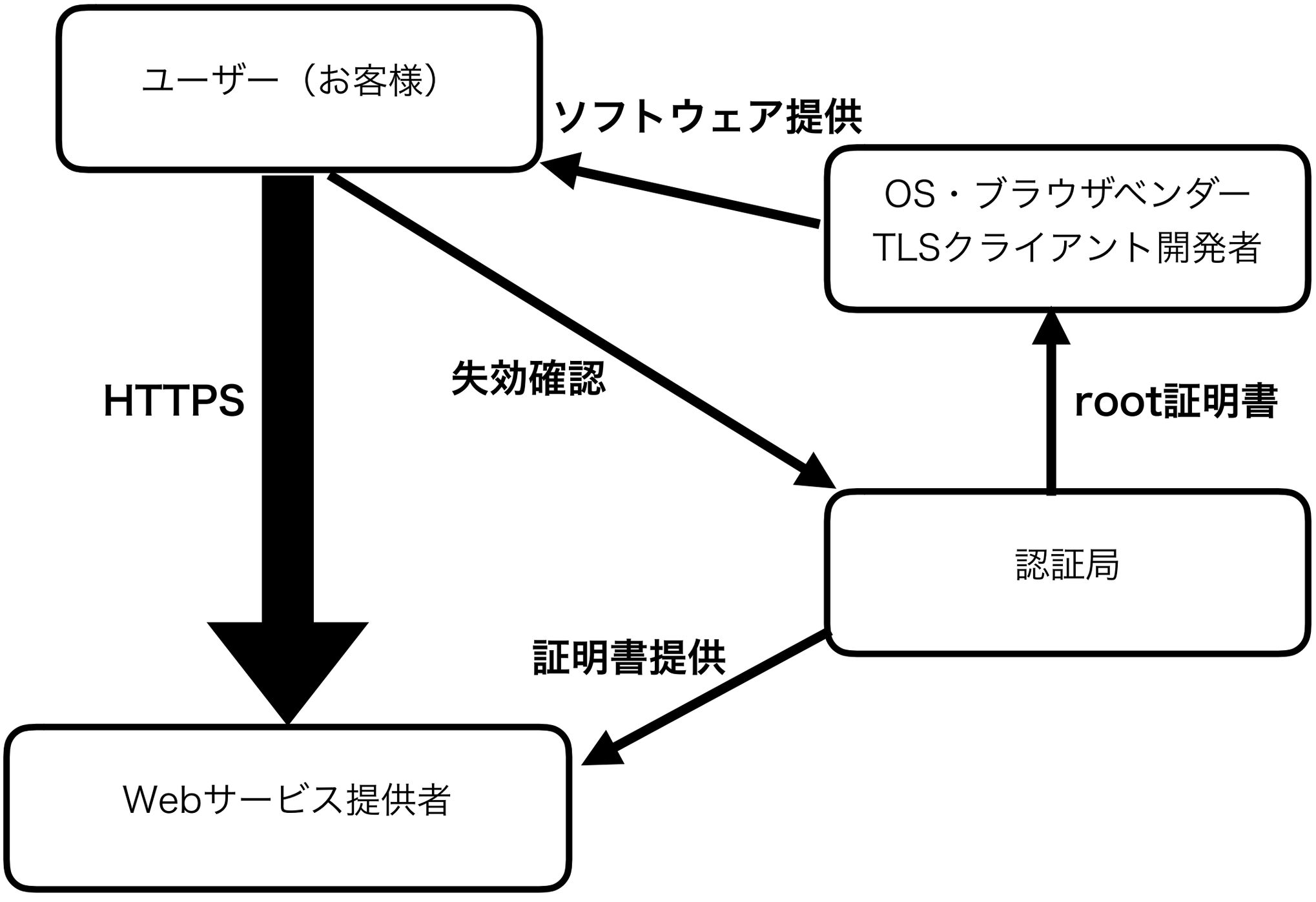
IETF QUICv1 の対象範囲外なので当面無理です。
やるなら自分でdraftと実装書いて提案しましょう。

QUICv1の仕様化が完了し、各種ブラウザーやOSS、
CDNサービスが出てくるまで待ちましょう。

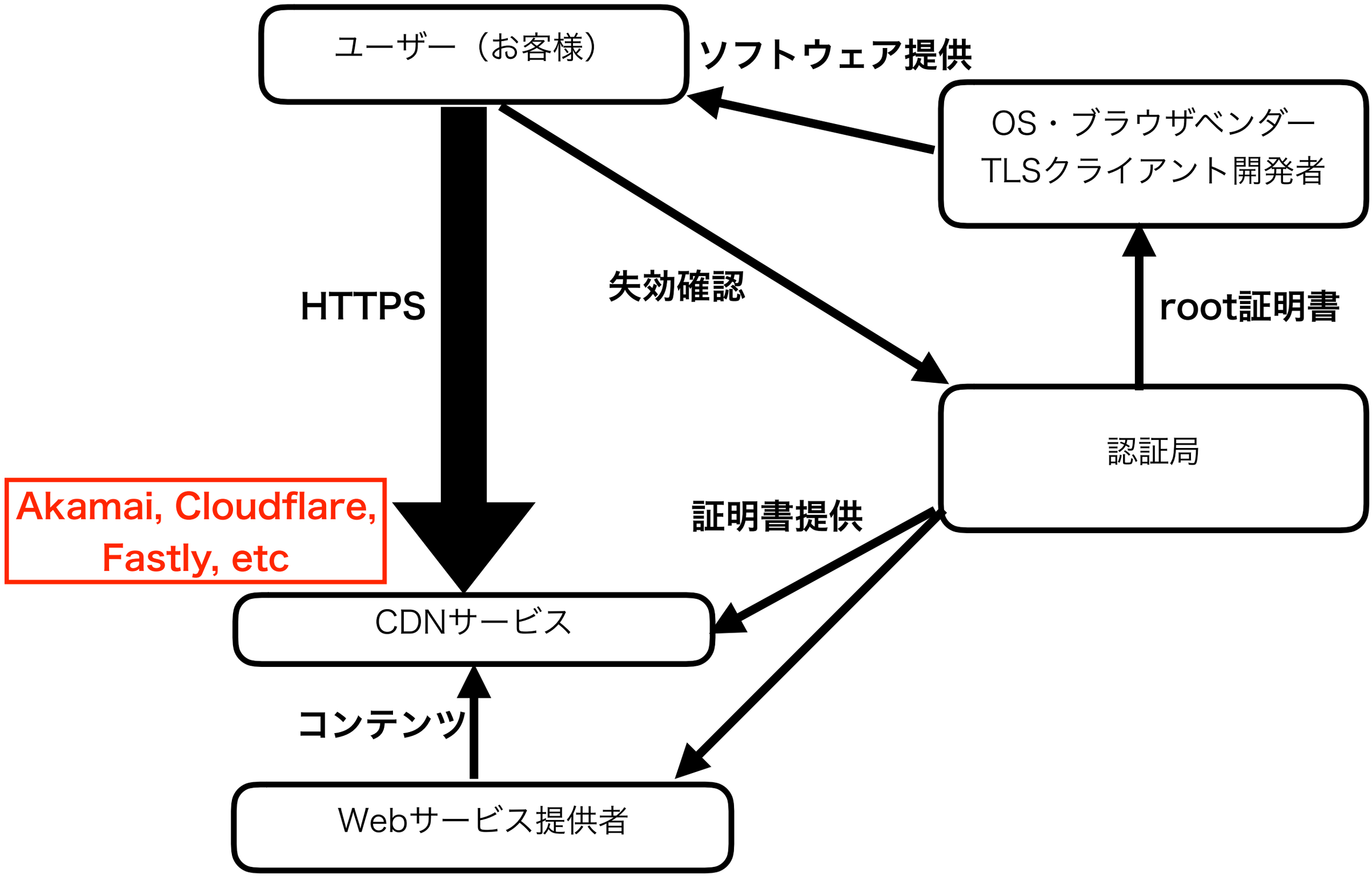
QUIC WGに参加してプロトタイプを作りましょう。

今後どうなっていくのか

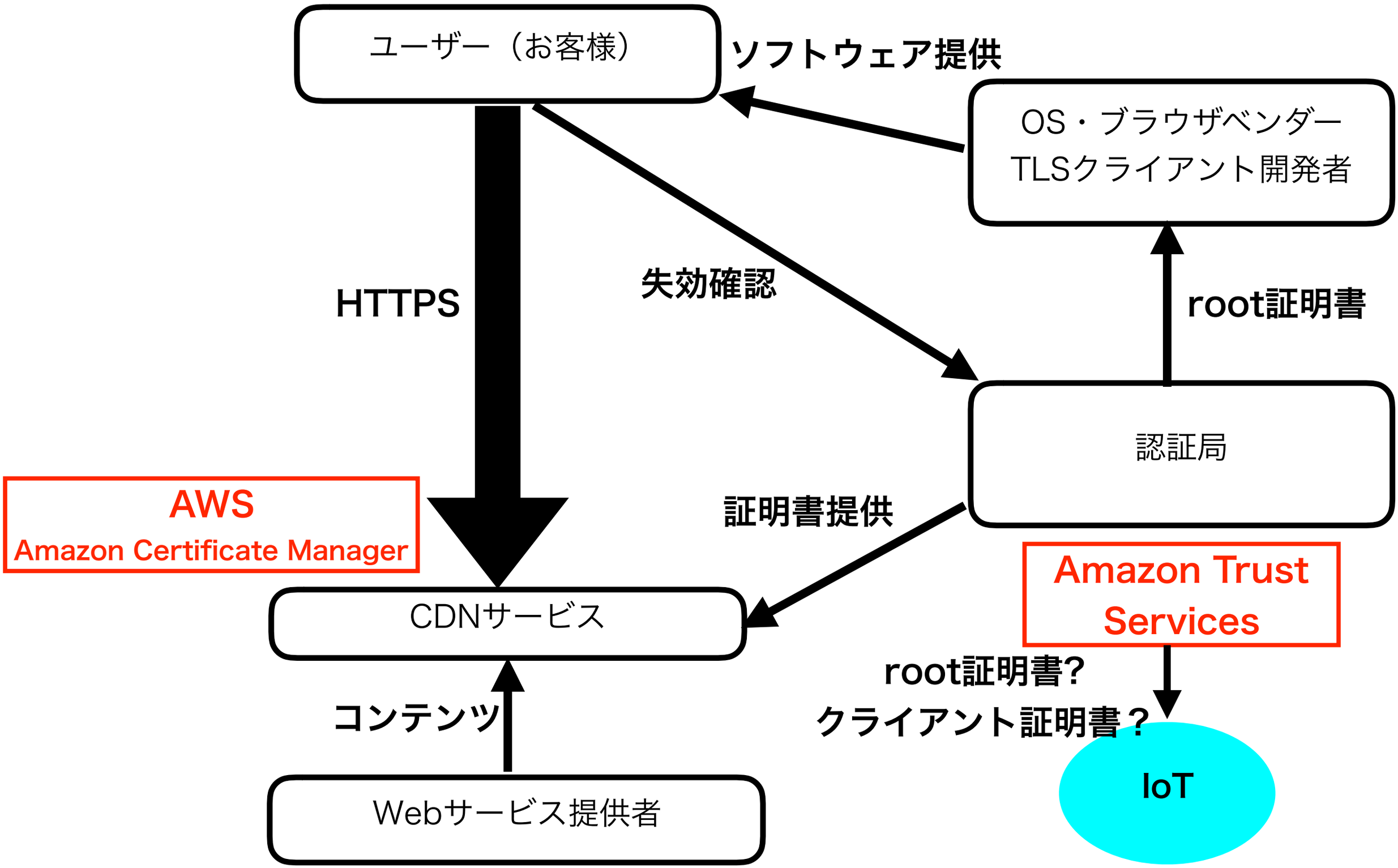
HTTPS everywhere 時代のステークホルダー



HTTPS everywhere
with Cloud CDN時代のステークホルダー



HTTPS everywhere
with Cloud CDN時代のAmazonの戦略



HTTPS everywhere
with Cloud CDN時代のGoogleの戦略