各種認証とその運用

名古屋大学 情報基盤センター
情報基盤ネットワーク研究部門
基盤ネットワーク研究グループ
嶋田 創

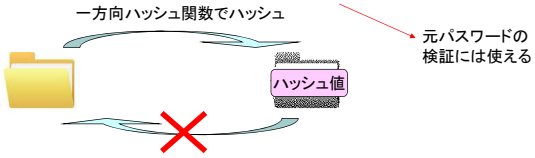
概要

- 認証を支える暗号技術
  - 一方向ハッシュ関数
  - 共通鍵暗号
  - 公開鍵暗号
  - 電子署名
- 典型的な認証と認証に対する攻撃
  - パスワード認証
  - 多要素認証
  - FIDO認証
  - 認証済み状態とその管理
- シングルサインオンや認証連携
- 認証に関する小ネタ

一方向ハッシュ関数(1/2)

(パスワード保存はハッシュ化して保存の厳守が基本)

- データから短い固定長の値(ハッシュ値)を出力する関数
  - データはハッシュ値に要約される
    - 例: MD5はどんなに長い入力データも128bitに要約
  - 入力データのわずかな変化でもハッシュ値は大きく変わる
- 一旦ハッシュ関数を通したら元のデータに戻れない
- サーバ側ではパスワードはハッシュ化して保存が基本
  - 元のパスワードに戻せない →漏れてもただちに影響は無い



一方向ハッシュ関数(2/2)

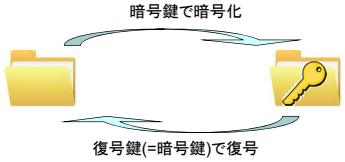
- 代表的な一方向ハッシュ関数
  - Message Digest 5 (MD5)
    - 1991年に策定された128bitのデータを出力するハッシュ関数
    - 既に寿命が来ており、MD5の利用はもうおすすめできない
  - Secure Hash Algorithm 2 (SHA-2)
    - 2000年に策定された256bitのデータを出力するハッシュ関数
    - 共通鍵暗号のDES/AESと同様に連邦情報処理標準に採用(SHS)
    - SHA-1(1995年/160bit)もあるが新規利用はおすすめでない(半分寿命)
    - 後継のSHA-3が2015年に策定完了
- 新しいものかつデータ長が長ければより安全だが、計算量などが増えるのが悩みどころ
- データ(ファイル)が改竄されていないことの証明にも利用
  - 正規のファイルのハッシュ値の公開や保存

一方向ハッシュ関数に関するセキュリティ

- 強衝突性: 同じハッシュ値を持つ複数のデータを作れる
- 弱衝突性: 指定されたデータのハッシュ値を持つデータを作れる
- 現像: 指定されたハッシュ値を出すデータを作れる
  - ただし、ハッシュ化されたパスワードにおいては、辞書攻撃で甘いパスワードは見つけ出される (これはどうにもしようもない)
    - パスワードの可能性のある文字列をハッシュ化→ハッシュ化されているパスワードと比較
- 上記の問題はハッシュ関数の寿命に影響
  - 暗号研究者が問題がないか一生懸命検証(攻撃方法の模索)をしている
  - 問題の大部分は「破るための時間が短くなる」レベル
    - ただし、コンピュータの計算能力の増大とともに、現実問題になることも

共通鍵暗号(1/2)

- よくある「データを鍵で暗号化して同じ鍵で復号」をする方法
- 特徴
  - 暗号化の鍵と復号の鍵は同じ
    - もしくは、暗号化の鍵から復号の鍵は容易に作れる
    - 古来からの暗号などもこのような形(シーザー暗号とか)
  - 離れた所で暗号通信路を構築するには鍵の受け渡しに工夫が必要
    - 特に「不特定多数の相手と暗号通信するウェブサービス」において
  - 基本的に、暗号化/復号の負荷は少ない(計算量が少ない)

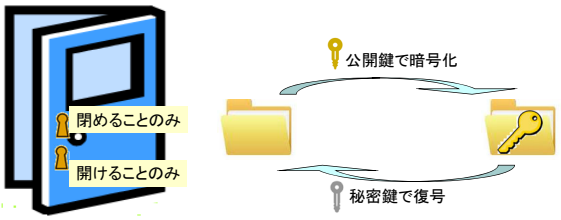


共通鍵暗号(2/2)

- 一定サイズのデータに対して暗号化処理を行なうブロック暗号が大多数を占める (対照: ストリーム暗号)
- 代表的なブロック暗号
  - Data Encryption Standard (DES)
    - 1977年に策定された連邦情報処理標準の暗号化手法
    - 鍵長は56ビット(異なる鍵で3重にかけるTriple DESもあり)
    - 既に安全ではない暗号化手法(残っていたら注意)
  - Advanced Encryption Standard (AES)
    - 2001年に策定されたDES後継の暗号化手法
    - 代表的な鍵長128bit, 192bit, 256bit(現状でどれも安全)
- セキュリティ対策側だけでなく、サイバー攻撃をする側も多用する
  - 例: ランサムウェアのファイル暗号化、トロイの木馬型マルウェアへの司令通信の暗号化、暗号化を利用したマルウェアのパッキング、など

公開鍵暗号(1/2)

- 共通鍵暗号の暗号鍵を安全に受け渡す時などに利用
- 鍵を、閉めること(暗号化)しかできない鍵と開けること(復号)しかできない鍵に分けることを考える(あくまでイメージ)
  - 暗号化しかできない鍵は公開する(公開鍵)
  - 復号しかできない鍵は自分で持つ(秘密鍵)



公開鍵暗号(2/2)

- 基本的に「暗号化されたデータは秘密鍵を知っていれば容易に復号可能だが、知らないと復号に非現実的な時間がかかる」数学的問題で作成される
- 代表的な公開鍵暗号
  - RSA
    - 素因数分解問題を利用
    - 代表的な鍵長(鍵長が長め): 512bit(非推奨), 1024bit(新規利用非推奨), 2048bit, 4096bit
  - ECDSA, ECDH
    - 楕円曲線(EC: Elliptic Curve)上の離散対数問題を利用
    - 代表的な鍵長:160bit(新規利用非推奨), 224bit, 256bit, 384bit
- 共通鍵暗号も含め、暗号方式と鍵長の寿命は[1]表3を参照

[1] IPA, "SSL/TLS暗号設定ガイドライン Ver. 3.0.1", 2020年7月.
https://www.ipa.go.jp/security/ipg/documents/ipa-cryptrec-gl-3001-3.0.1.pdf

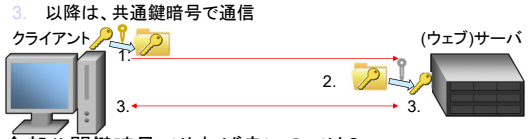
余談: 量子コンピュータ実用化よる暗号の解読ってどうなのよ?

- RSAやEC系は量子コンピュータに弱いと言われている
  - というか「量子コンピュータが実用化すれば…」な槍玉にされるぐらい
- 嶋田の個人的な主観
  - 現状で実用化の量子ビット数($10^{3\sim4}$)+αでは、現在公開鍵暗号で標準利用されているビット長の暗号鍵を解くのは無理($10^{6\sim9}$ほど必要)
  - 量子ビット数を増やすほど量子状態の維持が幾何級数的に難化
  - とりあえず、ビット長を長くすれば当分大丈夫では? (すごいブレイクスルーが無い限り)
- もちろん、対量子コンピュータ暗号(ポスト量子暗号)の研究もだいぶ前から行われている
  - 米国NISTがポスト量子暗号の標準化を開始(2017年)
  - 一般的な暗号の標準化には5年ぐらいかかるので、そろそろ…
  - OpenSSHは2022/4から格子暗号系とEC系の組合せが基本に[1]

[1] https://japan.zdnet.com/article/35186176/

一般的な公開鍵暗号を使った共通鍵暗号の暗号鍵の受け渡し

- 動作(例: HTTPSによる通信)
  1. 通信開始側が乱数より生成した共通鍵を公開鍵で暗号化して送信
  2. 通信を受け取る側は秘密鍵で送られてきた共通鍵を復号
  3. 以降は、共通鍵暗号で通信



- 全部公開鍵暗号でやれば良いのでは?
  →公開鍵暗号の処理速度は共通鍵暗号と3桁ほど違う
  →全て公開鍵暗号で処理していたら遅すぎ
- 共通鍵暗号の暗号鍵は通信セッションごとに使い捨て

暗号等はかならず解かれたりするもの

- 絶対に解読されない暗号や、衝突を作られないハッシュ関数は無い
- 解読までに非現実的な時間があれば良い
- コンピュータの進歩のよって解読までの時間は短くなる
  →暗号には寿命があると考え、適時交換する
  - 例: RSAの鍵は512bit→1024bit→2048bit→4096bitと延びている
  - 通常は新規サーバや新規サービスに移行時に変更
  - サービス途中で対応終了する事例も
    - 例: SHA-2非対応端末は2016/8/24にてモバイルSuica対応終了
    - Google Chromeだと鍵長が短いと鍵マークが緑にならなかったりする
- 暗号の寿命は3スライド前のIPA「SSL/TLS暗号設定ガイドライン」参照

暗号の強度と鍵空間

- いくら情報等の保護方法を準備しても、肝心の暗号鍵/ハッシュ値が同じになる可能性が高ければ意味がない
- 一般的に、鍵やハッシュ値のビット長で評価される
  - 暗号鍵/ハッシュ値のとりうる値は鍵長やハッシュ値長をnとすると2^n
    - 鍵長やハッシュ値長が長いほど処理の負荷が重くなる
  - 例: 128bit = 3.4×10^{38}、192bit = 6.2×10^{57}、256bit = 1.6×10^{77}
  - もちろん、アルゴリズムによっても強度は変わる
    - 一般的に、より短い鍵長と同程度の強度に落ちる方向になることが多い
    - 利用期間途中で脆弱性が見つかって「より短い鍵長と同じ強度しか無い」という落ちも
- 鍵やハッシュ値がとりうる値の空間を鍵空間と呼ぶ
  - 可能な限りこの空間一杯を使う
  - 個人のパスワードの生成においても、「鍵空間を広く使う」という意識は持ったほうが良い

概要

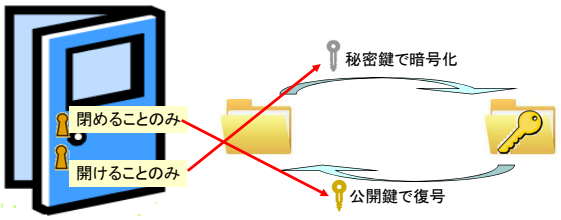
- 認証を支える暗号技術
  - 一方向ハッシュ関数
  - 共通鍵暗号
  - 公開鍵暗号
  - 電子署名
- 典型的な認証と認証に対する攻撃
  - パスワード認証
  - 多要素認証
  - FIDO認証
  - 認証済み状態とその管理
- シングルサインオンや認証連携
- 認証に関する小ネタ

電子署名

- 一般的な署名の用途
  - 文章に対して署名した人が責任を持つことを認める
  - 例:「文章 = 契約書」となって契約を承認する
  - 例:「文章 = 会計報告」となって会計報告に責任を持つ
  - 例:「文章 = 推薦書」となって推薦したことに責任を持つ
- 電子署名の用途
  - 基本的に同じだが、「文章」がデータであったりサーバになったり…
  - 署名の検証は一般的な署名よりも厳密にできる
    - 署名の流用やデータ等の改ざんはできない
  - 例: www.example.comは確かにXXサービスのウェブサーバ
  - 例: データに対して電子署名をつけて署名後の改ざんを検出可能に

電子署名の原理

- 公開鍵暗号の秘密鍵と公開鍵の役割が逆になる
- 署名を記したい人は秘密鍵で暗号化し、検証したい人が公開鍵で復号して検証するものと考える
  - 厳密には、ファイルのハッシュ値を秘密鍵で暗号化し、復号結果と手元のファイルのハッシュ値が一致すれば検証完了

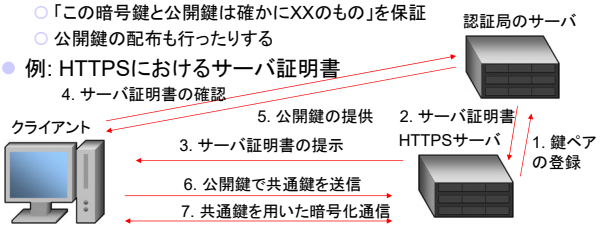


電子署名の保証

- 偽者が「秘密鍵と公開鍵の組」を生成して「この鍵で検証されたぞ」と言い放題ならば意味がない
  - 例: SSL/TLS証明書のオレオレ証明書は信用してはいけない

→信頼できる所に本人証明してもらう

- 認証局(Certification Authority)
  - 「この暗号鍵と公開鍵は確かにXXのもの」を保証
  - 公開鍵の配布も行ったりする
- 例: HTTPSにおけるサーバ証明書



証明書チェイン

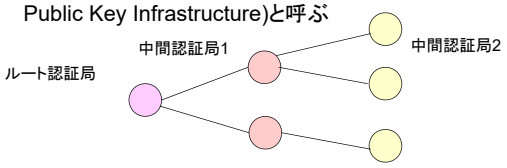
- 次は「その認証局は信頼できるのか」となる

→いくつか大元になるルート認証局が存在する

  - たいていはOSやブラウザに登録されている
- ルート認証局に仕事が集中してしまう

→中間認証局が準備されていて階層を作っている

  - 中間認証局は1つ上の認証局で保証される
- このような種々の認証局の基盤を公開鍵暗号基盤(PKI: Public Key Infrastructure)と呼ぶ



ブロックチェーン(1/2)

- 信頼の大本を担保するルート認証局を作れない場合の電子署名みたいなデータ保障はどうする?
  →分散保持と多数決による確認で保障できるのでは?
- 基本的なブロックチェーンの動作
  - 一定の区切りのデータを追加するごとに新たな保障を作る
    - 例: ハッシュ値をベースに、前のデータ系列のハッシュ値と新たなデータをあわせてハッシュ値を生成
    - 「過去の保障が正しいか(過去部に改竄が入っていないか)」も要検証
  - 上記を不特定多数のコンピュータで実行して信頼を確保
  - 新規に保障すべきデータを追加したい人は、不特定多数のコンピュータ網にデータをブロードキャスト
- 偽データの追加をできたりしないのか?
  - 不可能ではないが、ブロック追加時の承認手続きで極力排除

ブロックチェーン(2/2)

- 近年話題が(少し)減った仮想通貨の基幹をなす技術
  - データ(通貨の取引)追加時の新たな保障の作成や過去の保障の検証にインセンティブを与える(新たな仮想通貨を与える)
  - 現在の通貨はそれを発行している所(国など)の信用をもとに(相対的な)価値が決まる →各仮想通貨のシステムへの信用が価値となる
  - NFT(Non-Fungible Token)なる物にも使われている
    - トークンに紐付けられた物などの所有権を記録するという主張(なお、合法的に紐づけられたかどうか保証されなかったりすることも)
- 欠点: 利用すればするほどデータ系列が長くなる
  - 履歴の検証にかかる時間がどんどん長くなる
  - 細かな決済データがどんどん増えていく用途には向かない
- 契約の公的証明の代用とかになら有用そう
  - 公証人制度よりも手間やコストをかけたくない場合とか
  - 前述のハッシュ関数等の寿命話があるので期限が来たら要更新

概要

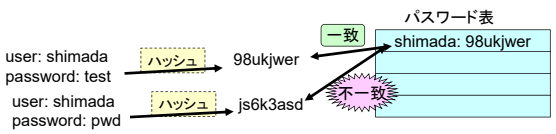
- 認証を支える暗号技術
  - 一方向ハッシュ関数
  - 共通鍵暗号
  - 公開鍵暗号
  - 電子署名
- 典型的な認証と認証に対する攻撃
  - パスワード認証
  - 多要素認証
  - FIDO認証
  - 認証済み状態とその管理
- シングルサインオンや認証連携
- 認証に関する小ネタ

認証の基本

- 認証の3要素: 識別、認証、認可
- 識別: 誰を認証しようとするのか?
  - 例: ユーザIDで識別
- 認証: どうやって識別した本人であることを確認するか?
  - 例: 当人しか知らないはずのパスワード等で認証
- 認可: 認証後に何を許可するのか?
  - ユーザの属性によっては閲覧できない情報もある(例: 教員と生徒)
  - ユーザの属性によっては実行できる処理に差があったりする(例: Windowsの制限ユーザと管理者ユーザ)
- 単一の情報サービスだと複雑ではないが、最近では認証連携によるシングルサインオンなどで複雑に
  - 例: Googleの認証情報でXXサービスに認証

パスワード認証

- ユーザIDとパスワードで認証する一般的な方法
- 動作
  1. パスワード登録時は一方向ハッシュ関数を通した結果をパスワード表に登録(ユーザIDとともに登録)
  2. 認証時には入力パスワードを一方向ハッシュ関数に通す
  3. 双方の結果が一致すれば認証OK
- パスワード表が漏れても、基本的にただちに影響は無い
  - パスワードがある程度複雑ならば、総当り攻撃などで破られるまでの時間は稼げる

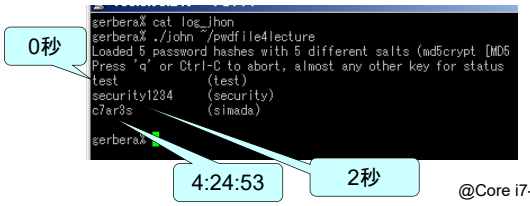


どのように認証や暗号化を破ってくるか?

- 防衛のためには破られ方を理解しておいた方が良い
- 実際に破る試験(ペネトレーションテスト)の実施もよくある
  - セキュリティ監査企業などがこういう仕事もうけています
  - この方面の専門家はホワイトハットハッカーなどと呼ばれます
- ただし、くれぐれもブラックハット側に陥らないように
  - まずは自分の管理下のシステムで実験しましょう
    - 近頃は、仮想マシン等でノートPC上にシステムを作るのも容易です
  - 他と競いたくなったら、CTFその他のコンテストや競技に出ましょう
    - セキュリティ関係企業その他が初心者向けから上級者向けまでいろいろコンテストや競技会を開いています
    - 余談: (日本の)ホワイトハットハッカーの芽を摘む行為は非常によろしくないと思う
      - 海外のホワイトハットハッカーもちゃんと育成して守りを固める
        →じゃあ、海外の悪人はどこを狙う?
      - ホワイトハットハッカーが開発する防御技術の知財が無い国は、知財取引でどういう扱いにをされるか?

(漏れたハッシュ化済み)パスワード表への総当り攻撃

- 色々なツールが出ています
  - 管理者側が弱いパスワードを設定している人を探すのにも利用
- 例: John the Ripper
  - 総当りの対象とする文字を設定したりしてクラック
  - ユーザIDを元にした推測も可能
  - 後述する辞書ファイルもオプションで指定できます



@Core i7-2700

攻撃者がパスワード認証破りに使う技術(試行回数削減のための工夫)

- 人間側が鍵空間を狭めていることがよくある
  - 数字キーは打ちにくいのでアルファベットだけでパスワード
  - 日付をベースで決めた数字を挿入

→狭まった鍵空間を予測して攻撃

- 辞書攻撃: パスワードの一部に辞書にある文字を利用
  - 辞書: 英単語の辞書、よく使われるパスワード一覧、など
  - “辞書+α(数字、アルファベット、など)”の文字列による攻撃
  - 個人情報と関連した推測でカスタム辞書を作ることも
  - 流出パスワード集から敵対的学習で生成パターンを推測して生成[1]
- ユーザID側に対する辞書攻撃:
  - あるソフトウェアをインストールすると作られるユーザID
  - 臨時で作って消す忘れることが多いユーザID

[1] https://pc.watch.impress.co.jp/docs/news/1492292.html

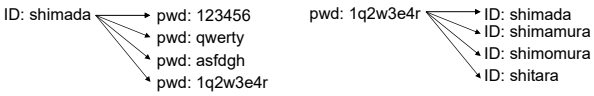
パスワードへのその他の攻撃(1/2)

- 個人情報と関連した推測
  - 電話番号とか住所の一部とか
- 他で漏れた認証情報の利用
  - 例: 2018/5のニコニコ動画における数百万件のログイン試行[1]
    - 流出したユーザIDとパスワードを他のシステムにとりあえず入れてみる
    - 特に電子メールアドレスをユーザIDとしている場合、ユーザIDは共通になってしまうことが多い
  - 漏れた認証情報は(一部は検証済みで)ブラックマーケットで売られているみたいで…
    - 最近だと、販売が確認されている物を警告してくれるサービスも多数
- 偽のパスワード入力欄への入力
  - 偽サイトへの誘導やproxyを利用した中間者攻撃など
  - パスワード入力欄(本物)の上に入力欄を作る悪性スクリプト(入りの広告など)

[1] http://news.nicovideo.jp/watch/nw3503730

パスワードへのその他の攻撃(2/2)

- 認証連携で認可と勘違いさせて全権限を得る(後述)
- パスワードスプレー攻撃
  - 従来はユーザID側を固定してパスワードを入れ替えて攻撃していたのを、パスワードを固定してユーザID側を入れ替えて攻撃
    - どれだけセキュリティ啓発しても安易なパスワードを設定する人は一定の割合(某先生の研究では約1%)で残るため
  - (利用者が限定される)システム管理者側にとって脅威
    - 一般利用者としてログインした後、権限昇格攻撃や他のシステムの攻撃を狙うとか



暗号化への総当り攻撃

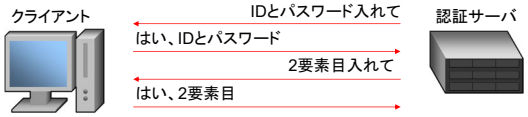
- 暗号化ZIPファイルのようにデータ形式が決まっていれば、いろいろと復号用パスワードを生成して入力するツールあり
  - いろいろと生成するパスワード候補は、パスワード破りと同様
- データ形式不明なら、以下を繰り返すプログラムで実施
  - 復号鍵を自動生成
  - 入力データの復号を試みる
  - 復号データを評価
    - 例: 一般的な文字コードから外れていないか?
    - 例: 既存の書式(Wordなど)に一致しているか?
- 解読速度: RC5-72bit(某暗号解読コンテストのクライアント)
  - CPU: Core i7-2600 で約1000万鍵/秒/1-core
  - GPU: Radeon HD 7750で約6億鍵/秒/512-core

概要

- 認証を支える暗号技術
  - 一方向ハッシュ関数
  - 共通鍵暗号
  - 公開鍵暗号
  - 電子署名
- 典型的な認証と認証に対する攻撃
  - パスワード認証
  - 多要素認証
  - FIDO認証
  - 認証済み状態とその管理
- シングルサインオンや認証連携
- 認証に関する小ネタ

多要素認証

- 認証時にIDとパスワード以外の要素を要求する認証
  - 普通は2要素だが、超重要システムだと3要素以上もありうる
- 重要: 2要素目は実質パスワードにならないようにすること
  - 「毎回変わらない固定された文字列」はパスワードに他ならない
  - ダメな例: 電話番号、誕生日、郵便番号、など
- 2要素目を後から入れるシステムが多いが、ID/パスワードの有効性確認に使えてしまう物は好ましくない
  - ID/パスワードを入れた段階で間違うと「間違いがあります」と言ってくるシステム



追加認証要素の例(1/2)

- マトリクス認証: 個人ごとに異なる文字換算表を渡して、手元で変換させる
  - 例: ほわはぬろをほち→しまだ
- ハードウェアトークン等による認証鍵生成
  - 個人ごとに異なるシード値を入れたハードウェアトークンを渡す
  - 時間や生成した回数などでシード値から毎回異なる数字を生成
    - 例: 1234(シード値)×202005180845 →162730(下位6桁)
    - 認証サーバ側は前後の値も許容したり
  - 最近ではアプリ版が主流(Google Authenticatorなど)

ち	り	ぬ	る	を	わ	か	
ろ	や	へ	に	そ	く	あ	い
わ	ゆ	ほ	ぬ	た	け	い	ろ
を	よ	ま	ね	ち	こ	う	は
ん	ら	み	の	つ	さ	え	に
゛	り	む	は	て	し	お	ほ
゜	る	め	ひ	と	す	か	へ
	れ	も	ふ	な	せ	き	と

マトリクス認証用の表

ハードウェアトークン

追加認証要素の例(2/2)

- ワンタイムパスワード(OTP: One Time Password)を別の認証を使うサービスを経由して受け取る
  - 事前登録したメールアドレスなど
  - SMS利用は超微妙(SMS狙いの攻撃、悪性SMSからの保護の問題)
- 生体認証
  - ただし、生体認証には「変更できない(しにくい)」という問題点もある
  - 保存された生体認証の特徴量が流出した場合、それをもとに認証用の顔なり指紋なりをGAN(敵対的生成ネットワーク)などで作れそう
    - 保存されているのは顔や指紋の写真そのものではない
    - もっと言うと、「見た目は元の生体情報と違うけど、なぜかシステムには元の生体情報と認証される」という物も作れそう
- 暗号鍵を入れたハードウェアトークン
  - 例: USBトークン、カードリーダー+カード
  - ソフトウェア的に認証サーバとの間で公開鍵認証を行う

多要素認証関係よもやま

- 利用者側の動きに応じて認証要素数を変える運用も
  - 1回多要素認証したら、一定時間はパスワードのみで認証OK
    - 強制ログアウト時間を短くしている情報サービスとかが多様
  - 特定のIPアドレス(組織の持つIPアドレス)からの接続ならばパスワードのみで認証OK
- ネットワーク話で出したように、多要素認証の2要素目を狙った攻撃(シード、SMS)も増えている点には注意
  - 安心しきってはいけない
  - ID/パスワードを突破した後に、ポップアップ型認証アプリに認証要求を繰り返し送る多要素認証疲労攻撃の被害も[1]
    - 私はポップアップで認証の可否を聞くタイプのアプリは「利用者が考える時間(手間ではあるが…)」を取らせないから大嫌い
    - 私はMicrosoft 365の認証もOATH-TOTPの6桁数字を使うよう設定変更している

[1] https://www.itmedia.co.jp/news/articles/2209/28/news050.html

エセ多要素認証

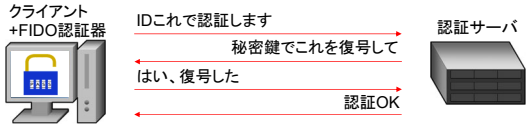
多要素認証に見せかけて実質1要素認証と変わらない

- パズル認証系
  - 「合わせ絵パズルを解いて」「上と同じだけクリックで色を変えて」
  - 認証の答えが同一ページにあるので、認証ですらない
    - 25年前に情報工学科でやった画像処理の演習レベルで自動化可能でロボット避けでも役に立たないレベル
- パスワードを2回入力させる
  - 長いパスワードを1回入れさせるのと安全性は変わらない
  - というか、勝手に個人情報を2つ目のパスワードにするのやめろ
- 認証時にOTP送付先メールアドレスを変更できる
  - 攻撃者からすると、自分のメールアドレスに変更すればOKなだけ
  - (ひょっとして「古いメールアドレスもう使えない」問い合わせ対策?)

FIDO(Fast IDentity Online)認証

認証の手間削減+サーバに認証情報を送らせない

- 公開鍵ベースの認証基盤
- FIDO認証器側で秘密鍵と認証先IDを管理
  - FIDO認証器を入れるデバイスはユーザ側で準備
  - FIDO認証器利用時に各種認証を求める(閉じた世界で各種認証)
- サーバ側にはIDと公開鍵を登録(FIDO認証器を通じて)
- 認証時には、サーバ側がIDに対応した公開鍵で乱数を暗号化して送りつける(正しく復号して送り返せたらOK)

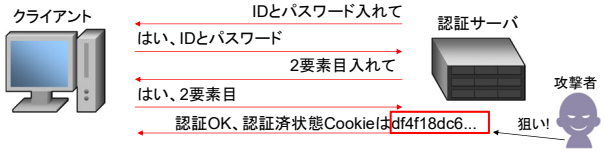


認証通過状態の管理

- 1回認証通ったのにサービス利用のたびに認証を要求されるのは不便
  →認証通過状態のIDを付与してそれをもとに利用を許可
- ウェブサービスではCookieが一般的(前回の話)
- ただし、あまりに長時間に渡って利用していない場合は再認証させた方が良い
  - サーバ側で認証通過状態IDと失効期限を設定
  - 失効期限までに再度サービスを利用した場合でも、適時IDを再設定する
- 認証通過状態IDを盗んで悪用するセッションハイジャック攻撃なるものも存在
  - 例: ブラウザのCookie保存ファイルから盗む、偽サイトに対してCookieを送出させる

多要素認証の増加で認証通過状態の窃取が進む?

- 多要素認証を使うサービスでも、再認証の面倒を省くために長く使える認証済状態Cookieを提供する所も多い
  - 酷い所になると、失効期限が(おそらく)無い所もある
  →攻撃者からすると認証済状態Cookieを盗んだほうが楽に
  - Cookie窃取だと、中間者攻撃系だけでなく、次回のXSS系も使える
- 本当にクリティカルなサービスならば、認証済状態Cookieは短時間で失効させる(=短時間で新しいCookieに更新)



認証過程を安全にするためには?

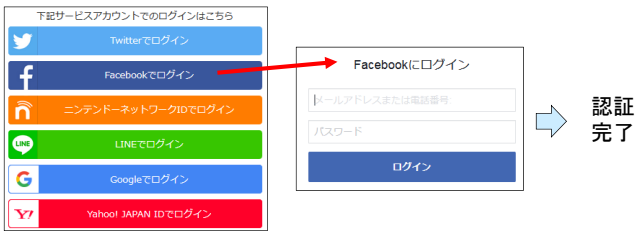
- 入力したユーザIDやパスワードはどのようにしてネットワーク上を安全に流れる?
  - ブラウザ側でハッシュ値にするのは意味はない
- HTTPSのように通信路を暗号化した上で認証へ
  - 逆に言えば、TLSを使っていないウェブページでパスワードを入力してはいけない(まだけっこう残っているが…)
  - HTTPSのように公開鍵認証で暗号化通信路を作る方法は多用される(Secured SHellやハードウェアトークン利用型認証など)

概要

- 認証を支える暗号技術
  - 一方向ハッシュ関数
  - 共通鍵暗号
  - 公開鍵暗号
  - 電子署名
- 典型的な認証と認証に対する攻撃
  - パスワード認証
  - 多要素認証
  - FIDO認証
  - 認証済み状態とその管理
- シングルサインオンや認証連携
- 認証に関する小ネタ

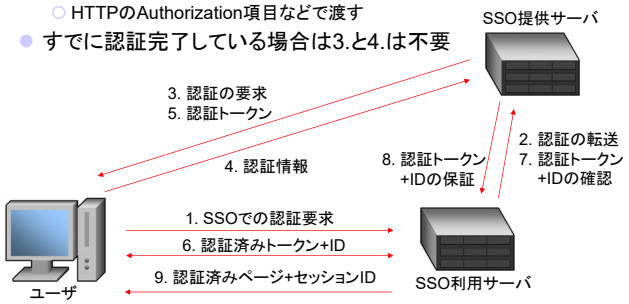
シングルサインオン(SSO)

- 複数のサービスで異なるID/パスワードを管理するのは大変
  →利用者の多いサービスと連携してそのIDで認証可能に
- シングルサインオン利用時の注意
  - クライアントが認証情報を中間で窃取する動きをしていないか?



正しく設計されたシングルサインオン

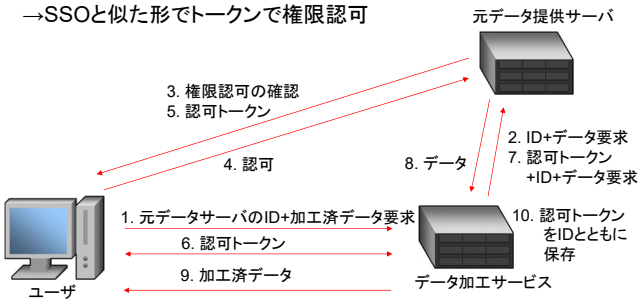
- SSO提供サーバと利用サーバ間は認証トークンを利用
  - トークンは利用サーバごとにユニークなID
  - HTTPのAuthorization項目などで渡す
- すでに認証完了している場合は3.と4.は不要



権限認可(権限移譲)による他サービスからのデータ利用

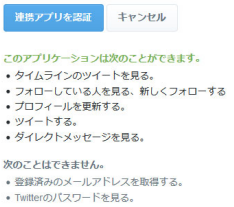
- 外部にデータを出す権限を認可する利用は多い
  - 例: Twitterクライアント(ウェブサービス、アプリ)

→SSOと似た形でトークンで権限認可



例: Twitterにおける権限認可

- 3種類に分けて権限を認可可能
  - 読むだけ
  - 読み書き可能(DMやアカウント情報は×)
  - 全権限
- 認可してもらう権限はクライアント側から申請
  - 必要な権限だけ認可されているか要確認
- OAuthという規格となっている
  - OAuth 2.0からSSOにも使えるようになった



SSOによる認証と権限認可の違いに注意

- 認証で問題ないのに、権限認可を間違って使われていることが無いか注意
  - 一応、「権限認可された = 権限をもらう先の認証は通っている」という解釈で、権限認可先の認証は通っていることは確認可能
  - でも、認証で問題ないのに権限認可を使うのは、不必要な権限をもらっていないか?
    - もらった権限を悪用しない保証は?

→ちゃんとSSOで認証されているか確認した方が良い

- 特に認証/認可のどちらでも使えるOAuth

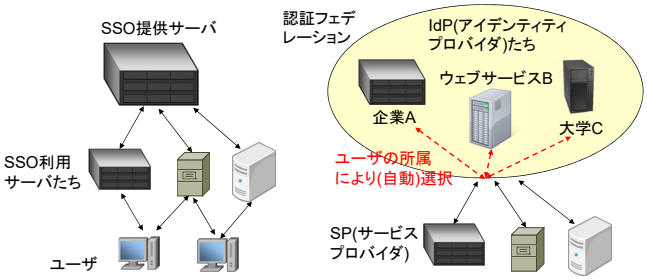
名古屋大学ID/機構アカウントでのSSO

- 名大IDはCASで各種情報サービスにSSOを提供
  - 名大IDとパスワードとOATH-TOTPによる多要素認証なSSO
- 素のCASを多少カスタマイズしてある
  - ロール(教員/学生/一般職員/常勤/非常勤などの身分や所属部局など)で認証の可否を設定可能
- 機構アカウントは契約したMicrosoft 365によるSSO
  - 機構アカウントとパスワードとMS認証アプリorOATH-TOTPによる多要素認証なSSO
  - ロール認証などは名大IDと同様に使える

認証フェデレーション

- シングルサインオンは認証情報を管理する所が1つ
  - SSO提供者側ををSSO利用サーバが明示する必要あり

→複数の認証サーバを連携させて1つのSSOに見せれば…

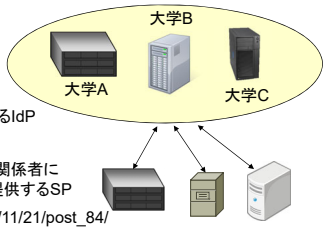


認証連携の例: 学認

- 学認:「学術認証フェデレーション」の略
  - 大学だけでなく公的な研究所なども参加
- SPは認められた利用者の属性(学生or教員)なども利用可能
- 将来的には様々な学割が電子化されるかも?
  - 学認で学生身分を確認する、高速バスのオンライン学割の試行[1]

GakuNin

http://www.gakunin.jp/



学術機関によるIdP

学生や大学関係者に
サービスを提供するSP

[1] https://www.well-net.jp/news/2017/11/21/post_84/

概要

- 認証を支える暗号技術
  - 一方向ハッシュ関数
  - 共通鍵暗号
  - 公開鍵暗号
  - 電子署名
- 典型的な認証と認証に対する攻撃
  - パスワード認証
  - 多要素認証
  - FIDO認証
  - 認証済み状態とその管理
- シングルサインオンや認証連携
- 認証に関する小ネタ

認証に関する小ネタ(1/3)

- パスワードを忘れたときのための「秘密の質問」の設定を要求してくる時にどうする? →長い乱数を突っ込もう
  - 「その答え、自分の他に知っている人が多い」な事例多数
    - むしろ、本人よりも詐欺師の方が正解率が高いという話も[1]
  - そもそも、認証を突破する手段が増えるのはセキュリティの低下
- そのサービスでアカウントを作って大丈夫か?
  - 個人情報は最低でもメールアドレスは要求される
  - そのサイトの利用状況なども悪用されたりする可能性は?
- 認証情報を管理/利用するアプリケーションはどこに認証情報保存している?
  - 端末側に保存されていると思いきや、クラウドサーバ側に保存されていることも…
  - 話題となった物: 家計簿アプリ、時間割アプリ

[1] https://security.srad.jp/story/22/04/25/1539229/

認証に関する小ネタ(2/3)

認証情報の漏れを確認するには?

- ログイン認証の日時やサービス利用履歴を確認する
  - 身に覚えのない日時に履歴がある

  →認証情報が漏れている、あるいは、セッションIDが漏れている
- 例: NUWNETの利用履歴

過去10日分のセッション情報

セッション強制ログアウト

※NUWNETから接続している場合は現在使用中のセッションもログアウトされてしまう場合がございますのでご注意ください。

建物名	開始日時	終了日時	利用時間	送信バイト数	受信バイト数
	2017/05/14 10:40:21	2017/05/14 20:07:57	09:27:36	190,470	281,165
	2017/05/14 10:34:12	2017/05/14 10:49:22	00:15:10	157,035	12,362
ES総合館	2017/05/13 19:32:26	2017/05/13 19:44:32	00:12:06	45,912	12,206
ES総合館	2017/05/13 19:21:49	2017/05/13 19:32:51	00:11:02	60,621	12,892

認証に関する小ネタ(3/3)

- 思った以上にパスワードをハッシュ化せずに平文で保存しているサービスは多い
  - 例: 宅ふぁいる便の平文パスワードを大量流出(2019/1)
  - 「パスワードを忘れた時」に設定中のパスワードが出てくるサービスには要注意
    - 「ランダム生成パスワードを提示し、自分で再設定させる」方が良い
- パスワードをハッシュ化する時のsaltって?
  - システムごとに入力の末尾に追加する文字(例: hs)
  - パスワード表の流出時に、saltも手に入っていないとパスワード破りができない