NPO 日本ネットワークセキュリティ協会
Japan Network Security Association

セキュリティ API の概要

アーキテクチャ、機能、暗号技術とアルゴリズム

東京大学先端科学技術研究センター
申吉浩

2004 年 8月 26日

このセッションの狙い

- 本セッションは、「セキュリティAPIに関する技術調査 Part 1」に基づき、以下を狙いとする
  - 10分でセキュリティAPIが必要な理由を理解する
  - 25分で専門用語に怯えない度胸をつける
  - 10分でセキュリティAPIの利用法のイメージを持つ
- Part 2以降のセッションでの専門的な話題を理解するための準備であり、暗号・セキュリティに関する専門知識を仮定しない…ほんの少ししか

JNSA

NPO 日本ネットワークセキュリティ協会
Japan Network Security Association

セキュリティAPIが必要な理由を理解しよう！

セキュリティAPIが必要となる二つの理由

JNSA

輸出の時に法律の規制を受けたくない

- 暗号機能が組み込まれたソフトは、外国為替及び外国貿易法、輸出令、外為令、貨物等省令による規制を受ける
- 国際的にもWassenaar条約による規制を受ける
- 暗号機能のみを“現地調達”することで規制を回避

使い易いAPIで暗号ライブラリを利用したい

- 小難しい暗号の詳細に煩わされたくない
- 一機能一関数が理想
- 利用者が多く、サンプルプログラム、トラブルシューティングノウハウが蓄積している環境が望ましい

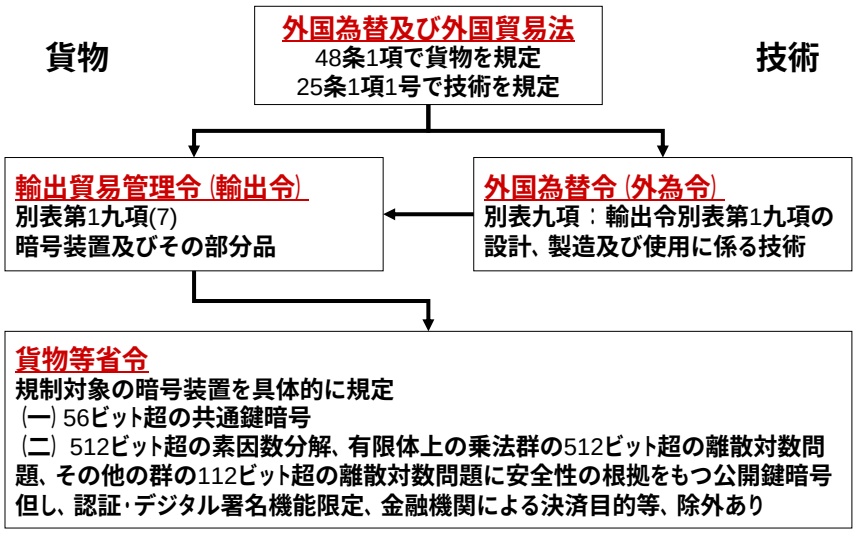
暗号に関連する日本の輸出規制

- 製品のみならず技術も規制の対象
- 「解読可能な」暗号以外は基本的に申請を義務付け
  - 56ビット超の鍵を利用する共通鍵暗号
  - 512ビット超の素因数分解、有限体の乗法群上で512ビット超、その他の群上で112ビット超の離散対数問題に安全性の基礎をおく公開鍵暗号
- 除外品目あり

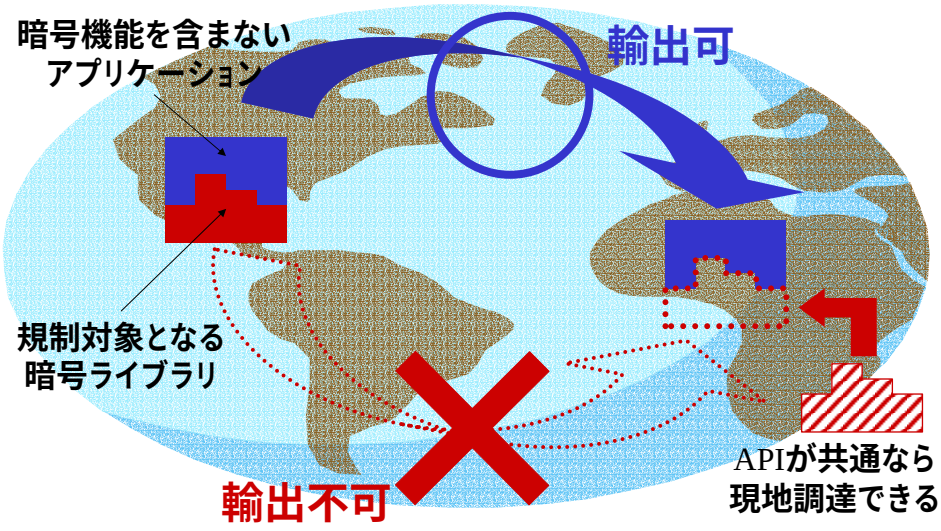
	地域 I	地域 II	その他
認証及びデジタル署名に限定	許可不要		
金融機関による決済目的			
個人情報保護のためのICカード			
特定加入者向け放送			
デジタルコンテンツ保護			
端末間通信を伴わない携帯電話及びコードレス電話			
マスマーケットプロダクト			
第1種一般包括許可の適用			許可要
上記以外			

地域 I：ｱﾙｾﾞﾝﾁﾝ、ｵｰｽﾄﾗﾘｱ、ｵｰｽﾄﾘｱ、ﾍﾞﾙｷﾞｰ、ｶﾅﾀﾞ、ﾁｪｺ、ﾃﾞﾝﾏｰｸ、ﾌｨﾝﾗﾝﾄﾞ、ﾌﾗﾝｽ、ﾄﾞｲﾂ、ｷﾞﾘｼｬ、ﾊﾝｶﾞﾘｰ、ｱｲﾙﾗﾝﾄﾞ、ｲﾀﾘｱ、大韓民国、ﾙｸｾﾝﾌﾞﾙｸﾞ、ｵﾗﾝﾀﾞ、ﾆｭｰｼﾞｰﾗﾝﾄﾞ、ﾉﾙｳｪｰ、ﾎﾟｰﾗﾝﾄﾞ、ﾎﾟﾙﾄｶﾞﾙ、ｽﾍﾟｲﾝ、ｽｳｪｰﾃﾞﾝ、ｽｲｽ、英国、ｱﾒﾘｶ合衆国
地域 II：ｱﾌｶﾞﾆｽﾀﾝ、ｲﾗﾝ、ｲﾗｸ、北朝鮮、ﾘﾋﾞｱ

輸出規制関連法の関係



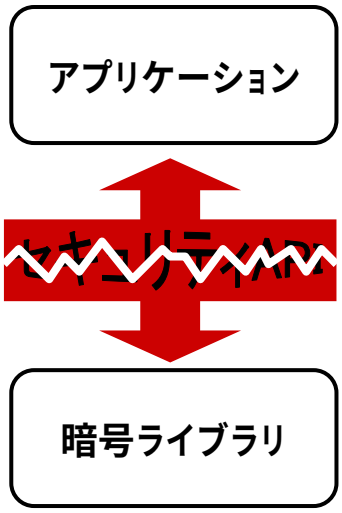
セキュリティAPIによる自由な輸出



使い易いセキュリティAPIとは

- セキュリティ機能の隠蔽
  - 暗号アルゴリズムなんか実装したくない
  - 小難しいセキュリティの勉強はいや
- 機能の集約
  - 一つの機能はできるなら一つの関数にまとめてよ
- 標準的な開発環境
  - まわりに経験者がいるとラクチンだ
  - やっぱりサンプルプログラムが分かりやすい
  - FAQやトラブルシューティングが充実してなくっちゃ

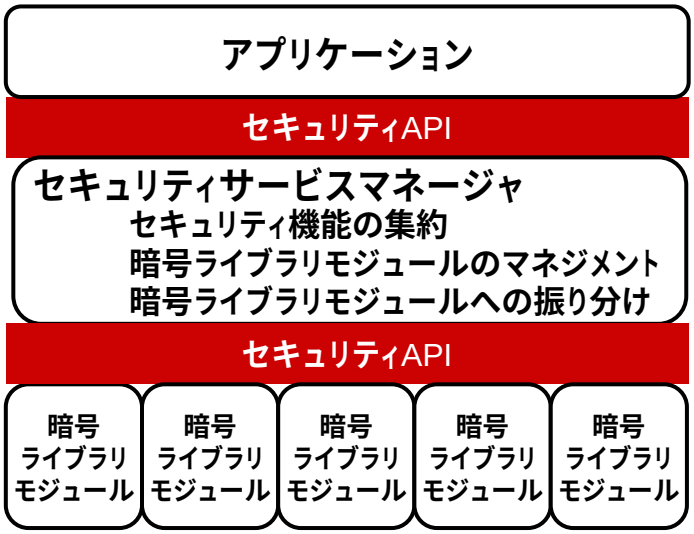
綱引き



使いやすさの観点からは…
プリミティブな機能を隠蔽し、高度な機能に集約していた方が使い易い

輸出の観点からは…
規制対象となるプリミティブな機能に絞って開発したほうがコストメリットがあり、モジュール化も容易で、拡張性がある

天地開闢 3層モデル



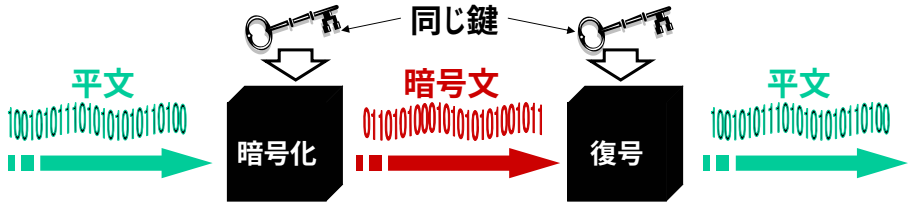
「鬼面人を威す」に負けるな！

専門用語なんて怖くない

JNSA

- **共通鍵暗号**、ブロック暗号、ストリーム暗号、DES、AES、CBC モード、鍵の全探索攻撃、差分攻撃、線形攻撃
- **メッセージ認証コード**（MAC）、CBC-MAC、HMAC
- **一方向性ハッシュ関数**、MD5、SHA-1、衝突と安全性
- **公開鍵暗号**、RSA、素因数分解問題、楕円曲線暗号、離散対数問題、OAEP
- **デジタル署名**、RSA署名、DSS、PSS

共通鍵暗号



- 暗号化と復号に同じ鍵を用いる暗号方式
  - 平文　暗号化するまえの生データ、ひらぶんと発音
  - 暗号文　内容が秘匿されたデータ
  - 暗号化　平文から暗号文を生成する処理
  - 復号　暗号文から平文を復元する処理、JISでは復号化といわない
  - 鍵　暗号化及び復号に必須のデータであり、鍵を保有する者だけが暗号化及び復号を実行できる。
  - 鍵長　鍵のデータとしての長さをビットで表した数、56ビットの鍵ならば、鍵のデータは7バイト、可能な鍵の数は2^{56}個となる。鍵長が大きければ大きいほど、強度が高まる（→鍵の全探索）。

ブロック暗号とストリーム暗号

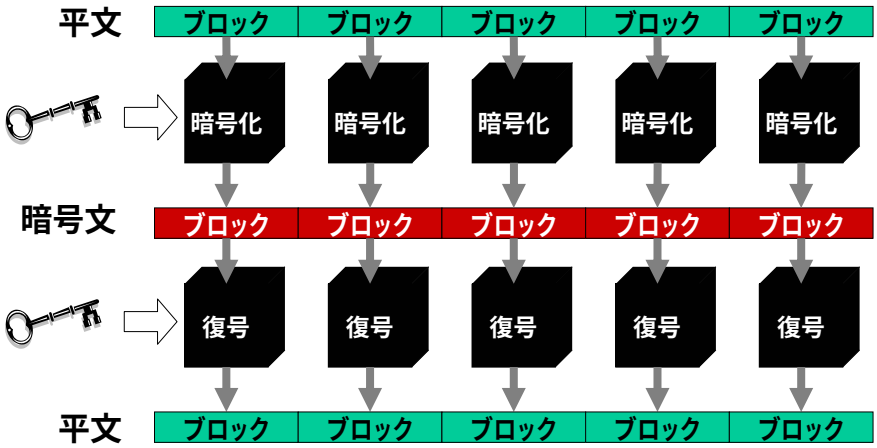
- 一度に暗号化･復号するデータの長さの違い
  - ストリーム暗号 計算機が処理し易い短い単位（1バイト等）
  - ブロック暗号　暗号強度を優先した長い単位（8バイト等）
  - 例えば、ビット置換（並べ替え）の複雑さは、8バイトブロックでは1バイトブロックの場合の10^{66}倍を上回る
- 一般的には、ブロック暗号が利用される

暗号方式	開発	鍵長	ブロック長	備考
DES	NIST 米国技術標準局	56 bit	64 bit	漸次、AESに代替
DES-EDE		112/168 bit	64 bit	Triple DES
AES		128-256 bit	128 bit	DESの後継
RC5	米RSA DSI社	1-256 bit	1〜256 bit	民間による開発

ブロック暗号の利用法 (ECBモード)

JNSA

長いデータはブロックに分割して逐次暗号化する



ナイーブなブロック暗号利用 (ECB) の危険

JNSA

平文	アリスは	ジョージ	を愛して	います
暗号文	a8f0b88cd90324bc	ef8d9b9cd378e89	c01200ef12546de	d12311023657efef
平文	ジョンは	ジョセフ	に借金が	あります
暗号文	b901c99de01435cd	f09eade489f9ad87	389efb8c32d9482	d12311f023657ef2

暗号文のブロックを入れ替えると偽造された平文が復号される

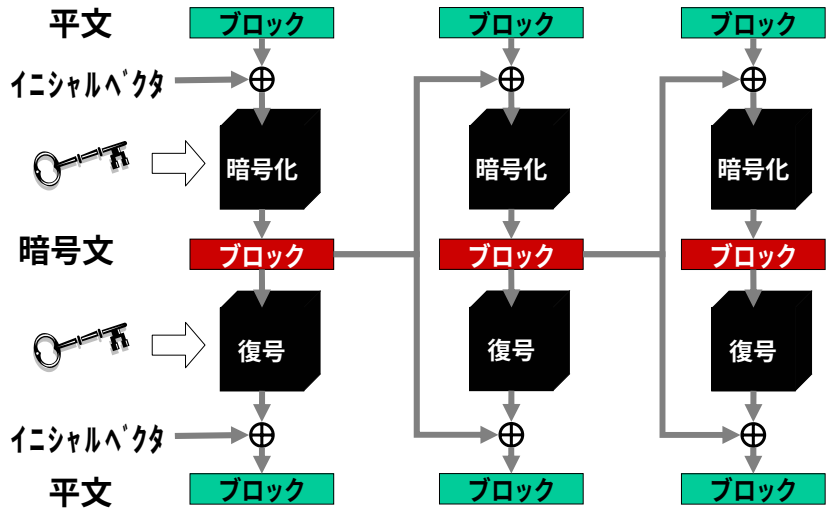
暗号文	a8f0b88cd90324bc	f09eade489f9ad87	c01200ef12546de	d12311023657efef
平文	アリスは	ジョセフ	を愛して	います

安全なブロック暗号利用　CBCモード

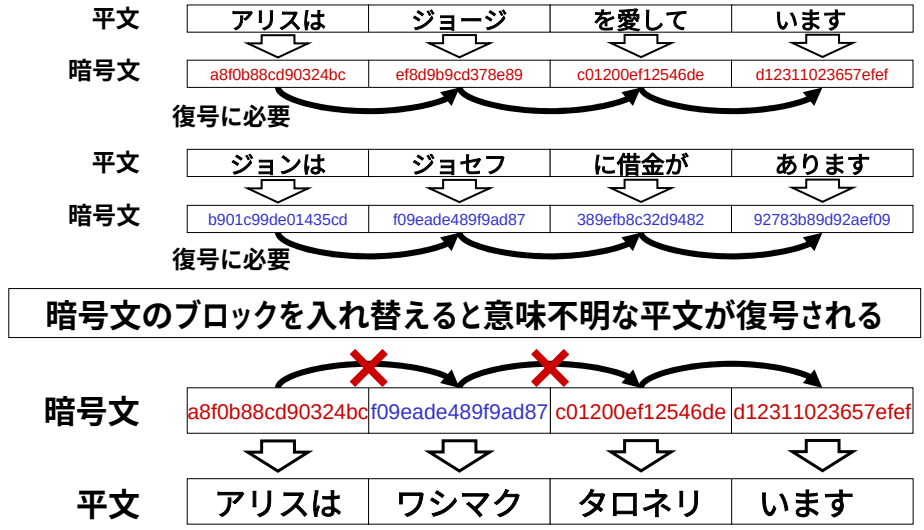
- Cipher Block Chaining.
- **暗号文ブロック間の連鎖** (Chaining)
  - **平文ブロックの暗号化はそれ以前のブロックに依存する**
  - **暗号文ブロックの復号には直前の暗号文ブロックが必要**
- **DESの利用モードとして米国技術標準局** (NIST) **が制定したのが最初**
- **ブロック暗号の利用モードとしては以下がある。**

モード	正式名称	特徴
ECB	Electronic Code Book	**ナイーブな利用法**
CBC	Cipher Block Chaining	**暗号化はそれ以前のブロックに依存**
CFB	Cipher Feed Back	**ブロック暗号をストリーム暗号として利用**
OFB	Output Feed Back	
PCBC	error-Propagating CBC	**暗号ブロックのエラーが連鎖**

安全なブロック暗号利用　CBCモード



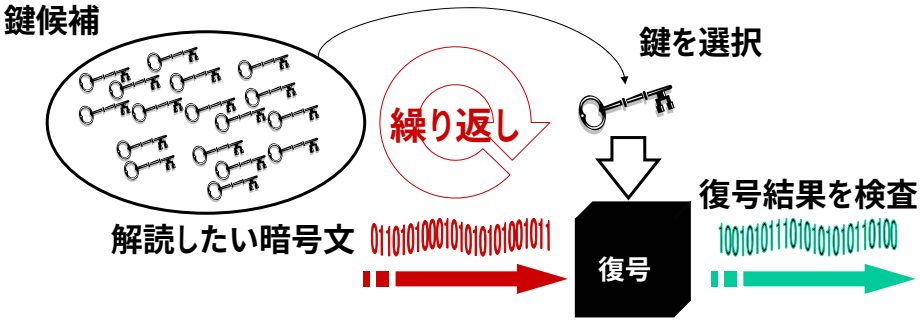
安全なブロック暗号利用　CBCモード



暗号文のブロックを入れ替えると意味不明な平文が復号される

鍵の全探索攻撃

正しい復号結果が得られるまで、鍵を取り替えて繰り返し検査
56ビット長の鍵を利用するDESでは、最大で2^{56}個の鍵を検査

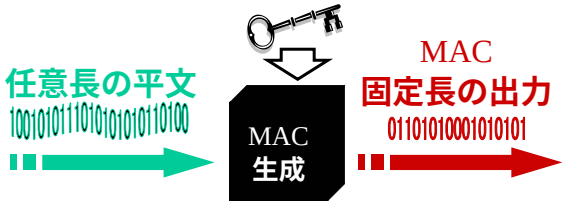


鍵長を増やす ⇒ 探索する鍵の候補が増える ⇒ 攻撃者の負担が増える

差分攻撃と線形攻撃

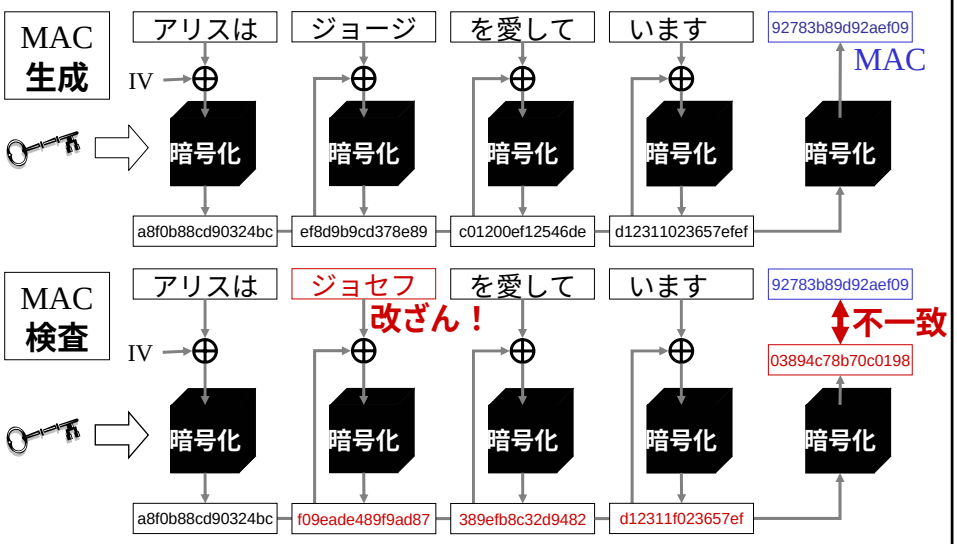
- 鍵の全探索攻撃より効率のよい攻撃法
  - 例えば、DESの場合では
    - 全探索攻撃 2^{56}個の鍵を探索
    - 差分攻撃　2^{47}個の（選択的）平文・暗号文ペアを解析
    - 線形攻撃　2^{43}個の平文・暗号文ペアを解析
- 差分攻撃と線形攻撃の発見の影響
  - DESではそもそも全探索攻撃が脅威であるので、差分攻撃・線形攻撃の影響は少ない
  - DESの後継であるAESは、差分攻撃・線形攻撃に対する耐性を持つよう設計されている

メッセージ認証コード（MAC）



- 鍵を使った変換という点では共通鍵暗号と同じ
- MACは固定長なので、MACから平文を復元できない。
- メッセージに添付して、メッセージの改ざんを検査するためのコードとして利用する。
- CBC-MAC とHMAC (Hash-based MAC)が利用される
  - CBC-MAC　最後のブロックの暗号化がそれ以前の平文に依存するCBCモードの特徴を利用
  - HMAC　一方向性ハッシュ関数（後述）を利用

CBC-MAC



一方向性ハッシュ関数

どんなに長い入力に対しても固定長のデータを出力する関数

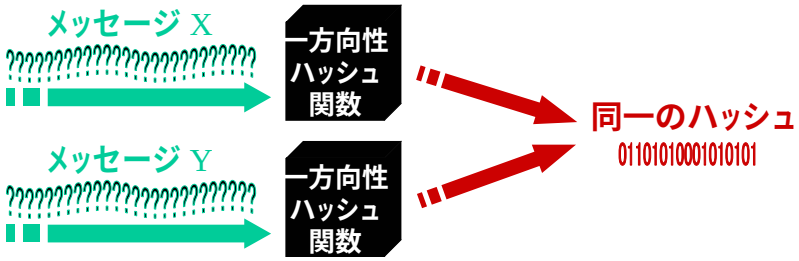
HMACをはじめ、多くの応用をもつ基礎的な関数

出力をハッシュ或いはメッセージダイジェスト

一方向性ハッシュ関数の性質（定義）

一方向性　与えられたハッシュをとるメッセージの発見は不可能

非衝突性　同じハッシュをもつ相異なるメッセージ（衝突、collision）の発見は不可能



一方向性ハッシュ関数

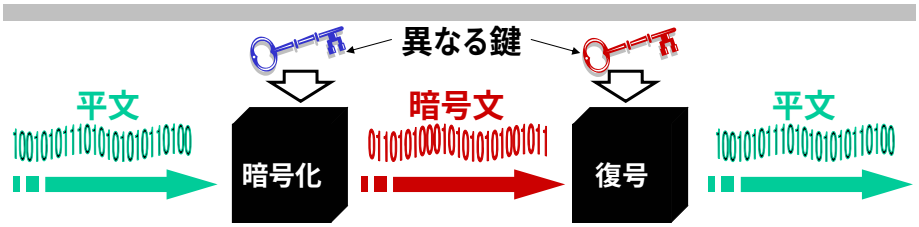
- 実用的にも理論的にも、暗号技術の重要な基礎をなす
  - 実用面では、メッセージ認証コード、安全証明つき公開鍵暗号、デジタル署名に利用される
  - 一方向性ハッシュ関数の存在は、現代暗号理論を支える最も基本的な「公理」のひとつである
- 実用的なアルゴリズムとしては、MD5とSHA-1が利用される

暗号方式	開発	ハッシュ長
SHA-1	NIST 米国技術標準局	20 byte
MD5	米RSA DSI社	16 byte

衝突（collision）と安全性

- MD5とSHA-1の衝突を計算するアルゴリズムの発見が、10年以上にわたって研究の焦点であった。
- MD5及び弱いバージョンのSHA-1（段数の少ない）で多くの衝突を計算するアルゴリズムがCrypto 2004 (August 17, 2004)において発表され、衝撃を与えた。
  - MD5 is fatally wounded; its use will be phased out. SHA-1 is still alive but the vultures are circling. (E.W.Felton, Princeton Univ., August 18, 2004)
    [対訳] MD5は致命傷を負い、消え行く運命にある。SHA-1はまだ息があるが、上空ではハゲタカが円を描いている。
- SHA-1の後継アルゴリズムの開発が急がれる

公開鍵暗号



- 1976年に存在を予言され、1978年にその存在（RSA暗号）が発見された新しい暗号方式
  - 暗号化と復号で異なる鍵を利用
  - 暗号化鍵から復号鍵を計算することは不可能
- 暗号化鍵を公開することで共通鍵暗号にない効用を実現する
  - 暗号化鍵を公開鍵と呼び、復号鍵を、秘密鍵、個人鍵、私有鍵、プライベート鍵などと呼ぶ

公開鍵ペア：公開鍵とプライベート鍵

- たとえ、公開鍵を「公開」してもプライベート鍵の発見が困難であるような鍵の構成法の発見が第一の関門
- 公開鍵ペアの構成には、次の二つの数学の問題の困難性が利用される
  - 素因数分解問題　RSA暗号の安全の基礎
  - 離散対数問題　楕円曲線暗号の安全性の基礎

素因数分解問題とRSA暗号

n が十分大きければ、n から素因数p,qの計算は困難

$$n = p \times q$$

Rabin暗号等では素因数分解問題の困難性を利用
nを公開鍵、(p, q)をプライベート鍵に設定

RSA暗号では、素因数分解問題と同値な問題を利用

$n=pq$ が十分大きければ、下の方程式を解くことは困難

$$ex = 1 \bmod (p-1)(q-1)$$

RSA暗号では、eを公開鍵、xをプライベート鍵に設定

離散対数問題と楕円曲線暗号

特定の有限群では、対数方程式を解くことは困難

$$b = a^{x}$$

但し、有限群は掛算と割算が定義された有限集合

連続な体（実数体、複素数体）では、$x = \log_a b$ により、容易に解ける事実と対照して、離散対数問題と呼ぶ。

離散対数問題が困難とされる有限体

1．有限体の乗法群

2．楕円曲線

bを公開鍵、xをプライベート鍵に設定

RSA暗号の安全性

RSA暗号の安全性は複数の仮説から構成される

素因数分解問題の困難性

||

公開鍵ペアの安全性

?

RSA暗号化関数の原理的安全性

||

実用上の安全性

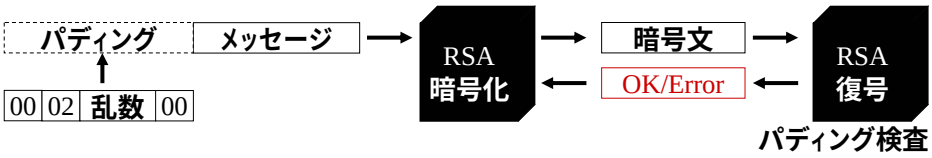
基本的な仮説であるが、素因数分解のアルゴリズムの改良が進んでいる

RSA仮説と呼ばれる：プライベート鍵の探索を経ない解読方法はあるか？素因数分解の困難性に帰着されると予想されている

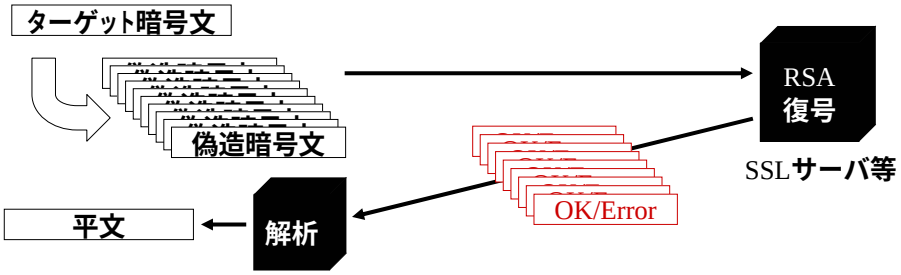
1998年に実用上の問題が発見され、OAEPによってRSA仮説に帰着されることが示された

RSA暗号の実用上の課題

RSA暗号の方式の規格PKCS #1においてパディングを規定



RSA復号のレスポンスを解析して、暗号文を解読することが可能



選択暗号文攻撃とOAEP

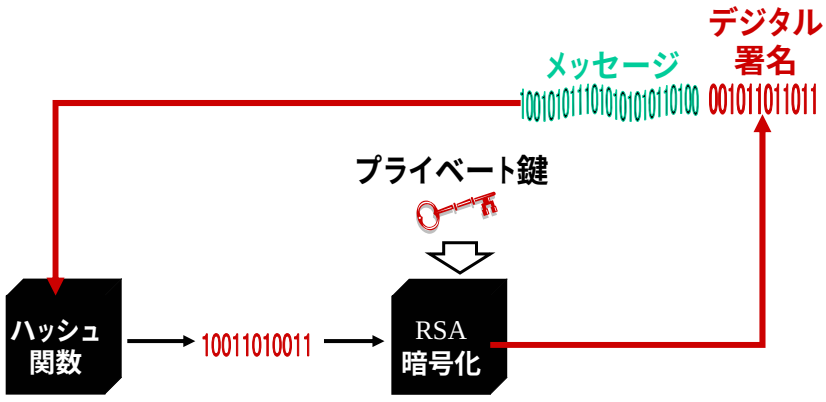
- 選択暗号文攻撃の可能性が現実化
  - 選択暗号文攻撃　　暗号文を選択的に生成、復号結果を解析することにより、ターゲットの暗号文を解読
  - 通信サーバ等を復号装置として利用
- OAEPによる解決
  - Optimal Asymmetric Encryption Padding.
  - 選択暗号文攻撃に対して、「証明つき」安全性を有するパディング方式
  - RSA仮説が正しければ、実用上安全であることを保証
  - PKCS #1に採用

デジタル署名

- 以下の目的でメッセージに添付されるコード
  - 改ざん検知
    - メッセージが改ざんされていないことを検査
  - 作成者認証
    - メッセージの作成者を特定
  - 否認拒否
    - メッセージ作成者が作成事実を否認することを許さない
- 実用アルゴリズム
  - 公開鍵暗号と一方向性ハッシュ関数を利用
  - RSA暗号を用いたRSA署名
  - 離散対数問題に基づくDSS署名（例、楕円曲線DSS署名）

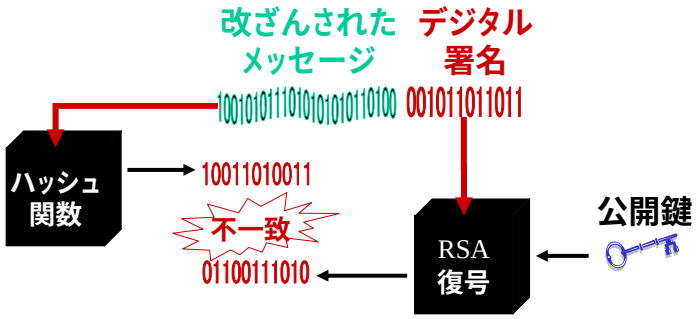
RSA署名

メッセージのハッシュをプライベート鍵で暗号化し、署名を生成



一方向性ハッシュ関数と署名の安全性

RSA署名の検証は、公開鍵でデジタル署名を復号することで実行
公開鍵とユーザとを対応付けることで、作成者認証と否認拒否を実現



一方向性ハッシュ関数に衝突が存在しないと仮定すると、
改ざんされたメッセージのハッシュとデジタル署名の復号は不一致

選択メッセージ攻撃とPSS署名

- 選択メッセージ攻撃
  - 公開鍵暗号に対する選択暗号文攻撃に相当する攻撃
  - メッセージを選択的に生成して、それに対するデジタル署名を解析することで、デジタル署名を偽造
- 選択メッセージ攻撃に対する安全性
  - DSS署名は証明可能な安全性を提供する
  - RSA署名は証明可能な安全性を提供しない
    - PSS署名はメッセージの符号化に乱数性を加えることにより、証明可能な安全性を実現
    - Pseudo-random Signature Scheme.

NPO 日本ネットワークセキュリティ協会
Japan Network Security Association

セキュリティAPI利用のイメージをつかもう！

セキュリティAPIの主な機能

- 証明書ハンドリング
  - 公開鍵証明書の検証
  - 公開鍵証明書の無効化の検証
  - 証明書の登録･検索･削除
- セキュア通信
  - SSL/TLSのハンドシェイク及びレコードレイヤプロトコル
- 鍵交換
  - 公開鍵に基づく鍵交換
- 暗号化/復号
- 一方向性ハッシュ（メッセージダイジェスト）
- デジタル署名
- メッセージ認証コード

プリミティブな暗号関数操作の例 共通鍵暗号による暗号化関数

```
//Example based on JCE
Cipher e = Cipher.getInstance(“DES/CBC/SSL3Padding”);
/*SSLにしたがってパディングした平文に対して、DESのCBCモードで暗号化/
復号を実行するCipherオブジェクトを生成*/
e.init(ENCRYPT_MODE, myKey);
/*生成したCipherオブジェクトを暗号化の目的で使用、鍵としてmyKeyを指定
*/
e.doFinal(clearText, offset, length, cipherText);
/*平文のバイト列clearTextを指定された方式で暗号化し、バイト列cipherTextに出力*/
```


プリミティブな暗号関数操作の例 共通鍵暗号による復号関数

```
//Example based on JCE
AlgorithmParameters iv;
iv =
AlgorithmParameters.getInstance(“DES/CBC/SSL3Padding);
iv.init(encotedIV);
//直列化されたイニシャルベクタをオブジェクトに変換
Cipher d = Cipher.getInstance(“DES/CBC/SSL3Padding”);
d.init(DECRYPT_MODE, myKey, iv)
//Cipherオブジェクトを復号目的で使用、復号のための鍵とイニシャルベクタ
を設定
d.doFinal(cipherText, offset, length, clearText);
//暗号文のバイト列cipherTextを指定された方式で復号し、バイト列
clearTextに出力
```

高レベルの暗号機能の例
SSLのハンドシェイクプロトコル

```
//Example based on JSSE
SSLSocketFactory factory =
      (SSLSocketFactory)SSLSocketFactory.getDefault();
SSLSocket secureSocket = (SSLSocket)factory.createSocket(host,
port);
//ホスト名とポート番号を指定してSSLSocketを生成
secureSocket.setEnableSesstionCreation(TRUE);
//生成したSSLSocketの上で新しいセッションを張ることを許可
String[] enabledCS = secureSocket.getEnabledCipherSuites();
//利用可能なCipherSuiteを事前に確認
BOOLEAN clientAuthNecessary = secureSocket.getNeedClientAuth();
//クライアント認証の必要の有無を事前に確認
secureSocket.startHandshake();
//ハンドシェイクプロトコルを実行
String[] supportedCS = secureSocket.getSupportedCipherSuite();
//サーバとの間で実際に合意されたCipherSuiteを取得
```


高レベルの暗号機能の例
SSLのレコードレイヤプロトコル

```
//Example based on JSSE
OutputStream os = secureSocket.getOutputStream();
//暗号化はアプリケーションから隠蔽され、送信メッセージはosを介して送られる
InputStream is = secureSocket.getInputStream();
//復号はアプリケーションから隠蔽され、受信メッセージはisを介して取得される
```