Seminar Kryptographie
WS 05/06

Angriffe auf RSA

Anika Frischwasser

Inhalt des Vortrags

- Wiederholung RSA
- PKCS #1 (RSA-Standart)
- RSAES-OAEP
- Chosen Ciphertext Attack
- Drei Algorithmen zur Berechnung von $m_0=(c_0)^d \bmod N$, wobei c_0 der abgefangene Chiffretext ist und m_0, die gesuchte Nachricht.
  - Versuche kleine Werte s_i zu finden, so dass $c_0(s_i)^e \bmod N$ dem Standard PKCS #1 entspricht und verkleinere Lösungsraum
  - Ähnlicher Angriff auf verbessertes RSAES-OAEP
  - Ausnutzung der Homomorphie-Eigenschaft von RSA

Wiederholung von RSA

1. Zufällige und stochastisch unabhängige Wahl zweier Primzahlen $p \neq q$, die etwa gleich lang sein sollten und Berechnung des Produkts $N = p \cdot q$.

   In der Praxis werden diese Primzahlen durch Raten einer Zahl und darauffolgendes Anwenden eines Primzahltests bestimmt

2. Berechne $\varphi(N) = (p-1) \cdot (q-1)$, wobei φ für die Eulersche φ-Funktion steht.

3. Wähle eine Zahl $1 < e < \varphi(N)$, die teilerfremd zu $\varphi(N)$ ist.

4. Berechne die Zahl d so, dass das Produkt $e \cdot d$ kongruent 1 bezüglich des Modulus $\varphi(N)$ ist, dass also $e \cdot d \equiv 1 \bmod \varphi(N)$ gilt.

Wiederholung von RSA

- Der öffentliche Schlüssel (public key) besteht dann aus

  * N, dem Primzahlprodukt sowie

  * e, dem öffentlichen Exponenten.

- Der private Schlüssel (private key) besteht aus

  * d, dem privaten Exponenten sowie

  * N, welches allerdings bereits durch den öffentlichen Schlüssel bekannt ist.

- p, q und $\varphi(N)$ werden nicht mehr benötigt und sollten nach der Schlüsselerstellung auf sichere Weise gelöscht werden.

- Berechne $c = m^e \bmod N$ als chiffrierte Nachricht.

Sicherheit von RSA

- Die Sicherheit von RSA basiert entscheidend darauf, dass die Primfaktorzerlegung des Produktes von N aus p und q nicht möglich ist.
- Es wurden inzwischen eine Reihe von Algorithmen entwickelt, die das Problem der Primfaktorzerlegung lösen. Diese sind je nachdem, welche Faktoren die zu untersuchende Zahl tatsächlich hat unterschiedlich schnell.
- Bei der Wahl der Primzahlen wird deshalb darauf geachtet, dass kein Algorithmus bekannt ist, der das Produkt der beiden Primzahlen schnell zerlegen kann.
- Man wählt die Primzahlen also so, dass möglichst alle bekannten Faktorisierungsalgorithmen in ihren schlechtesten Fall gezwungen werden.

PKCS (Public Key Cryptographie Standart) #1 v1.5

00	02	padd.string	00	data block

PKCS #1 Blockformat zur Verschlüsselung. Die ersten zwei Bytes sind konstant, die Länge des padding Blocks kann variieren(max 8)

Führendes 00 Oktet sichert die Konvertierung in eine kleinere Zahl als der Modulus

Padding String ist mindestens acht Oktets lang, umBrute-Force-Angriffe zu erschweren.

- Verschlüsselungsblock EB (k Bytes) mit EB = EB_1ll....ll Eb_k ist ***PKCS konform***, wenn gilt:
  - $EB_1 = 00$
  - $EB_2 = 02$
  - EB_3 bis EB_{10} sind nicht null
  - eines der Bytes von EB_{11} bis EB_k ist 00
- Chiffretext C ist PKCS konform, wenn die zu C gehörige Entschlüsselung PKCS konform ist.
- Definition enthält keine wichtigen Integritätschecks!

RSAES-OAEP (RSA Encryption Scheme with Optimal Asymmetric Encryption Padding)

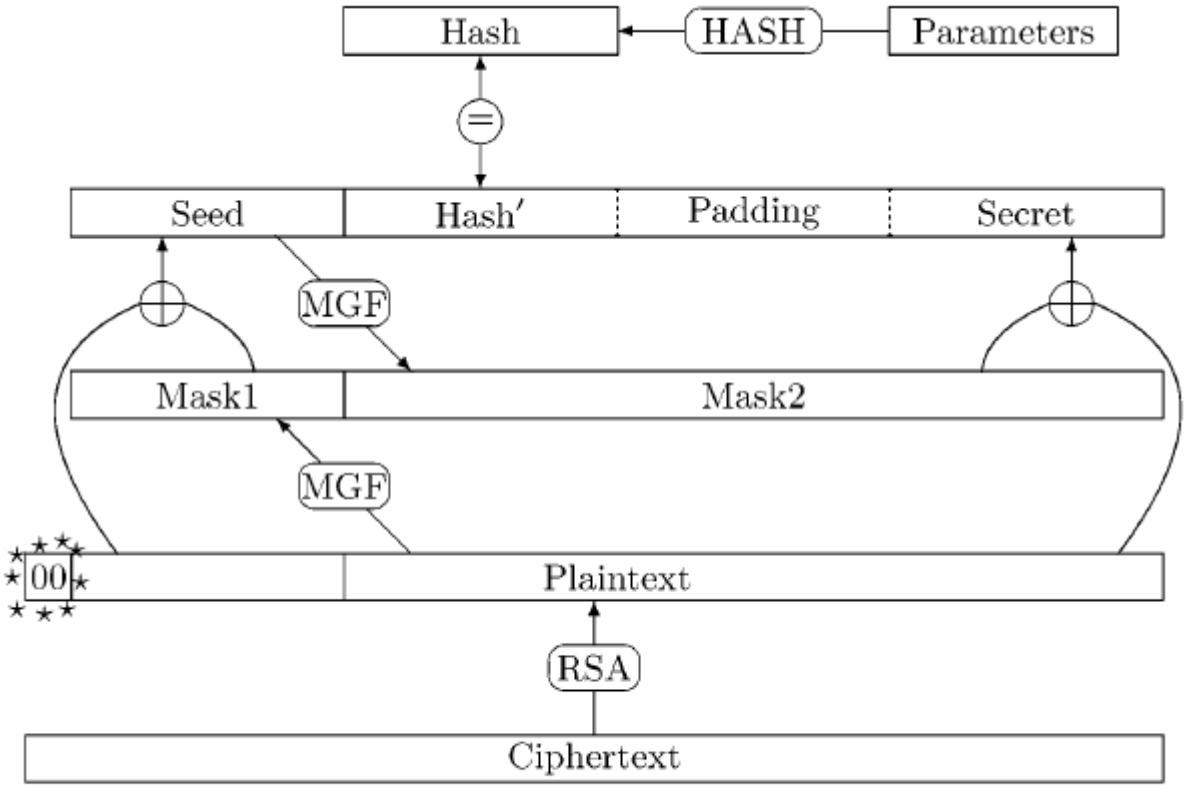

Fig. 1. RSAES-OAEP Decoding

RSAES-OAEP

- Integrität des Chiffretextes wird durch Vergleich des „unmaskierten“ Hash und eines unabhängig berechneten Hash der Parameter gewährleistet (und durch Prüfen des Padding)
- Entschlüsselung kann fehlschlagen bei
  - Integer-zu-Oktet-Übersetzung
  - während der OAEP-Entschlüsselung
- daraufhin gibt PKCS #1 v2.0 den Output „decryption error“ (in beiden Fällen) und bricht ab

Chosen Ciphertext Attack

- Der Angreifer kann sich die Chiffretexte aussuchen und erhält die zugehörigen Klartexte (CCA1)
- kann auch in adaptiver Form durchgeführt werden. Der Angreifer darf Chiffretexte in Abhängigkeit zuvor erhaltener Klartexte und den Ergebnissen von Zwischenrechnungen wählen (CCA2)
- RSA ist anfällig für CCA. Ein Angreifer, der $m = c^d \bmod N$ finden möchte, kann eine zufällige Zahl s auswählen und nach der Entschlüsselung von $c' = s^e c \bmod N$ fragen. Es ist leicht aus der Antwort $m' = (c')^d \bmod N$ auch die ursprüngliche Nachricht mittels $m = m' s^{-1} \bmod N$ zu erhalten.

1. Angriff auf RSA

- Annahme, dass der Angreifer Zugang zu einem Orakel hat, das für jeden Chiffretext beantwortet, ob der zugehörige Klartext PKCS konform ist oder nicht.
- Algorithmus verlässt sich auf die Tatsache, das die ersten beiden Bytes im Standard konstant und bekannt sind, wenn der Chiffretext als konform anerkannt wird.
- Als Vereinfachung sei $B = 2^{8(k-2)}$ ($L(N) = k$, Länge von N in Bytes) ist nun ms PKCS konform, dann impliziert dies $2B \leq ms \bmod N \leq 3B$
- M_i ist Menge von (abgeschlossenen) Intervallen, die nach erfolgreich gefundenem s_i berechnet wird so, dass m_0 in dieser enthalten ist

1. Angriff auf RSA (Algo)

1. Step: ***Blinding***

Gegeben c_0, wähle untersch., zufällige s_0 und prüfe mittels Orakel, ob $c(s_0)^e$ mod N PKCS konform ist.

Für das erste erfolgreiche s_0, setze

i. $c_0 \leftarrow c(s_0)^e \bmod N$

ii. $M_0 \leftarrow \{[2B, 3B-1]\}$

iii. $i \leftarrow 1$

1. Angriff auf RSA (Algo)

2. Step ***Suchen von PKCS konformen Nachrichten***

2. a) IF i = 1 THEN suche kleinstes, positives $s_1 \geq N/3B$, so dass $c_0(s_1)^e$ mod N PKCS konform ist

2. b) ELSE IF (i > 1 & #Intervalle in M_{i-1} mind. 2) THEN

suche die kleinste Zahl $s_i > s_{i-1}$, so dass $c_0(s_i)^e$ mod N PKCS konform ist

ELSE // i > 1 & $M_{i-1} = \{[x,y]\}$

wähle kleine Zahlen r_i, s_i so dass

$r_i \geq 2(ys_{i-1}-2B)/N$ und $(2B+r_iN)/y \leq s_i \leq (3B+r_iN)/x$

bis $c_0(s_i)^e$ mod N PKCS konform ist

1. Angriff auf RSA (Algo)

3. Step ***Eingrenzen der Lösung***

Nachdem s_i gefunden ist, setze

$$M_i \leftarrow U_{(x,y,r)}\{ [\max(x, \lceil (2B+rN)/s_i \rceil), \min(y, \lfloor (3B-1+rN)/s_i \rfloor)] \}$$

für alle $[x,y] \in M_{i-1}$ und $(xs_i-3B+1)/N \leq r \leq (ys_i-2B)/N$

4. Step ***Berechnung der Lösung***

Wenn M_i nur noch ein Intervall der Länge eins (d.h. $M_i = \{[x,x]\}$) enthält, dann setze

$$m \leftarrow x(s_0)^{-1} \bmod N$$

und gib m als Lösung von $m_0 = c_0^{\,d} \bmod N$ aus.

Sonst setze $i \leftarrow i-1$ und GOTO Step 2

Analyse des Algorithmus

- Wahrscheinlichkeit, dass eine zufällige Zahl $0 \leq m \leq N$ PKCS konform ist (Ereignis P), beträgt $0{,}18*2^{-16} < Pr(P) < 0.97*2^{-8}$
- Warum funktioniert's?
  - zz: m_0 ist immer in den M_i enthalten:

    Induktion über i:

    IA: Ist m_0 PKCS konform $\Rightarrow 2B \leq m_0 \leq 3B-1 \Rightarrow m_0 \in M_0$

    IS: $(i-1 \rightarrow i)$

    Sei $m_0 \in M_{i-1} \Rightarrow$ es ex. Intervall $[x,y] \in M_{i-1}$, so dass $x \leq m_0 \leq y$. Ist $m_0 s_i$ PKCS konform, dann ex r so, dass $2B \leq m_0 s_i - rN \leq 3B-1$ und folglich $xs_i - (3B-1) \leq rN \leq ys_i - 2B$. Ausserdem gilt $(2B+rN)/s_i \leq m_0 \leq (3B-1+rN)/s_i$

    $\Rightarrow$ Nach Def der M_i ist $m_0 \in M_i$

Analyse des Algorithmus

- Komplexität:

  Für $Pr(P) = 0{,}18 * 2^{-16}$ und $k = 128$ (entspricht 1024-Bit Modulus)

  erwartet man ca 2^{20} zu wählende Chiffretexte um einen erfolgreichen Angriff zu führen.

  (Pr(P) gilt unter der Annahme, dass die s_i unabhängig voneinander sind)

- Ein Modulus der Bitlänge 8k-7 würde den Angriff leichter machen, dann sind nur ca. 2^{13} Texte auszuwählen.

  $2^{20} = 1048576$ und $2^{13} = 8192$

Was sind Schwachstellen?

- Wird keine (oder an zu später Stelle) Autentifizierung gefordert, so kann Eve als Alice auftreten und Bob als Orakel „nutzen“
  - laut PKCS keine feste Stelle für Autentifizierung festgelegt
- Wenn der Angreifer vom Empfänger detaillierte Informationen über die Art des Fehlers erhält, wenn die Nachricht nicht PKCS konform ist.
- Zum Beispiel unterschiedliche Fehlermeldungen, wenn Nachricht zu gross oder falsches Format

Was sind Schwachstellen?

Timing Attack

- zuverlässiger Integritätscheck oft nur Teil der Signatur, nicht der Verschlüsselung

1. Sei $m \equiv c^d \bmod N$
2. IF m nicht PKCS konform

   THEN Fehler
3. ELSE verifiziere Signatur von m
4. IF Signatur falsch

   THEN Fehler

   ELSE OK

- Nicht möglich Chiffre zufällig zu generieren, so dass die zugehörige Nachricht eine korrekte Signatur ist
- Aber: Nachricht die 2., aber nicht 3. passiert kann erzeugt werden!
- Mit Messung der Serverantwortzeit kann ein Angreifer erkennen, ob das gesendete c PKCS konform war oder nicht.
- unterscheiden nur zwischen Ausführung von Entschlüsselung oder Entschlüsselung+Verifikation

Praxis-Test

Getest wurde eine Implementation mit Orakel, welches beantwortet ob Nachricht PKCS konform ist oder nicht.

Test mit 512-Bit und 1024-Bit Schlüssel

⇨ Algo brauchte zwischen 300.000 und 2.000.000 gewählten Chiffretexten

2. Angriff auf RSA

- Annahme, dass der Angreifer Zugang zu einem Orakel hat, das für jeden Chiffretext beantwortet, ob $m \equiv c^d \bmod N < B$ gilt.
- Als Vereinfachung sei $B = 2^{8(k-1)}$ (L(N) = k, Länge von N in Bytes)
- Annahme, dass $2B < N$ (ist normalerweise erfüllt)
- Angriff läuft ähnlich zum 1. Angriff
- Initial ist $m \in [0, 2B)$
  - ➔ da $m < B$, ex immer ein Vielfaches von m, das in einem Bereich der Breite B liegt, zB $\forall i \exists f : f*m \in [in, in + B)$

2. Angriff auf RSA

1. Step ***Teste Vielfache von 2, 4, 8, … 2^i… solange das Orakel „≥ B“ sagt***

1.1 Wir wissen: $m \in [0,B)$, teste $f_1=2$

1.2 Also ist $f_1 m \in [0,2B)$. Sende $(f_1)^e c \bmod N$ an das Orakel

1.3a IF „< B“ → impliziert $f_1 m \in [0,B)$, also ist $2f_1 m \in [0,2B)$. Setze $f_1 \leftarrow 2f_1$ und GOTO 1.2

1.3b IF „≥ B“ → impliziert $f_1 m \in [B,2B)$ für ein (nach Konstruktion) gerades $f_1 \Rightarrow (f_1/2) \in [B/2,B)$

2. Angriff auf RSA

2. Step ***Starte mit f_2, so dass $f_2 m$ kleiner als n+B ist für maximal mögliches m. Erhöhe f_2 bis*** „< B“

2.1 Wir haben: $(f_1/2)m \in [B/2,B)$, sei $f_2 = \lfloor (N+B)/B \rfloor * (f_1/2)$

2.2 Also $f_2 m \in [N/2,N+B)$. Teste f_2 im Orakel

2.3a IF „≥ B“ → impliziert $f_2 m \in [N/2,N) \Rightarrow (f_2+f_1/2)m \in [N/2,N+B)$

Setze $f_2 \leftarrow f_2+f_1/2$ und GOTO 2.2

2.3b IF „< B“ → impliziert $f_2 m \in [N,N+B)$

Während f_2 in den Iterationen über 2.3a grösser wird, so wächst auch die untere Schranke (überschreitet evtl auch n, wenn $f_2 = \lceil 2N/B \rceil * f_1/2$

2.3b tritt ein und Step 2 terminiert (max $\lceil N/B \rceil$ Orakel-Anfragen)

2. Angriff auf RSA

3. Step ***Teste Vielfache f_3, die einen Bereich für $f_3 m$ von der Breite 2B. Jede Orakel-Antwort halbiert den Bereich***

3.1 Wir haben: $f_2 m \in [N,N+B)$, haben f_2 und Bereich $[m_{min},m_{max}]$ für mögliche Werte von m $m_{min} = \lceil N/ f_2 \rceil$, $m_{max} = \lfloor (N+B)/f_2 \rfloor$ und $f_2(m_{max} - m_{min}) \approx B$

3.2 Wähle f_{tmp} ,so dass der Bereich von $f_{tmp} m$ annähernd 2B ist

3.3 Wähle eine Grenze in+B nahe des Bereichs von $f_{tmp} m$, $i = \lfloor f_{tmp} m_{min}/N \rfloor$

3.4 Wähle f_3, so dass $f_3 m$ an in+B angrenzt und teste es im Orakel.

$f_3 = \lceil iN/ m_{min} \rceil \Rightarrow f_3 m \in [iN,iN+2B)$ – f_3 ist approximativ äquivalent zu f_{tmp}

3.5a „≥ B“ → impliziert $f_3 m \in [iN+B,iN+2B)$. Setze $m_{min} \leftarrow \lceil iN+B/ f_3 \rceil$ und GOTO 3.2

3.5b „< B“ impliziert $f_3 m \in [iN,iN+B)$ Setze $m_{max} \leftarrow \lfloor iN+B/ f_3 \rfloor$ und GOTO 3.2

Analyse des Algorithmus

- Jede Antwort des Orakels wählt entweder obere oder untere Hälfte des f_3m-Bereichs und halbiert damit den Bereich der mögl. Werte für m.
  - evtl verengt sich der Bereich, in dem m liegt, zu einer einzelnen Zahl (diese ist dann m)
  - Ist $N \geq B$, so liegt $f*m \in [0,B) \cup [N,2B)$ und die Entscheidung welche f zu testen sind wird komplizierter (CCA2 aber immer noch möglich)
- Komplexität:
  - Steps 1 und 3 halbieren den Wertebereich von m mit jeder Iteration (benötigen $\log_2 B = 8(k-1)$ Orakel-Anfragen)
  - Step 2 braucht $\lceil N/B \rceil$ Orakel-Anfragen (muss ≤ 256 sein – im Mittel ist es die Hälfte dieser Zahl)

Praxis-Test

- RSA-Moduli sind typischerweise 1024 Bit
  - ⌈ N/B ⌉entspricht (128,256]
  - Step 2 braucht ca 100 Anfragen
- für einen 1024 Bit RSA-Schlüssel benötigt der Angriff ca 1100 Anfragen (für 2048 ca 2200)

Logs und weiteres

- Auch wenn das System in den Protokoll-Antworten Details vermeidet, kann man genauere Fehlerbeschreibungen in den Logs finden (z.B. „Integer too large“, „decoding error“, z.B. bei PKCS #1 v2.0 aus den Unterroutinen)
  - diese Details machen einen Angriff erfolgreich
  - Logs sind leichter zugaenglich als Zugang zu einem privaten Schluessel zu erhalten.

- Basis des Angriffs ist der Zugang zum Orakel, das „m<B“ beantwortet. Erfolgreich wird ein Angriff jedoch durch andere Details.

Homomorphie-Eigenschaft von RSA

- Für den öffentlichen Schlüssel bestehend aus (e,N) eines RSA Systems gilt $(m_1)^e (m_2)^e \bmod N = (m_1 m_2)^e \bmod N$.
- Das sagt man kann eine Nachricht $m_1 m_2$ verschlüsseln, ohne m_1 und m_2 zu kennen.
- Den gleichen Angriff kann man auf das RSA-Signaturverfahren starten. Und aus den Signaturen $(m_1)^d \bmod N$ und $(m_2)^d \bmod N$ kann man eine gültige Signatur $(m_1 m_2)^d \bmod N$ einer Nachricht $m_1 m_2$ bilden, ohne den privaten Schlüssel d zu kennen.
- Das Problem, dass sich bei diesem Angriff stellt ist, dass die Nachrichten m_1 und m_2 einen Sinn ergeben, die Nachricht $m_1 m_2$ allerdings nicht. Das liegt daran, dass die Redundanz der Nachrichten durch die Multiplikation zerstört werden. Hat man eine Nachricht die nicht redundante Werte beinhaltet, sollte man Redundanz einfügen um diesem Angriff zu entgehen.

3.Angriff auf RSA (Teilbarkeitsangriff)

- Definition 1
  - N mit $N = \Pi_{i=1}^{k} p_i$ mit p_i prim ist y-glatt, wenn $p_i \leq y$ fuer $1 \leq i \leq k$

- Definition 2
  - N mit $N = \Pi_{i=1}^{k} p_i$ mit p_i prim und $p_i \geq p_{i+1}$ heisst semi-glatt bzgl. y und z, wenn $p_1 \leq y$ und $p_2 \leq z$. Wir sagen N ist (y,z) semi-glatt

3.Angriff auf RSA (Teilbarkeitsangriff)

- Proposition
  - Sei $B < N$, $x \in \{0, ..., B-1\}$ und a so, dass $1 \leq a \leq N/B$ mit ggT(a,N)=1. Sei $y \equiv x/a \bmod N$, dann gilt

    $y \in [0,B) \Leftrightarrow a \mid x$
- Proposition sagt:

  Teilt p die geheime Nachricht m_0?
- Angriff versucht nach der Proposition grosse glatte Teile der Nachricht m_0 zu finden und den Rest der Nachricht mittels anderer Methoden, z.B. Offline-Brute-Force-Suche

3.Angriff auf RSA (Teilbarkeitsangriff)

- ***Parameter***: festes S als Schranke des glatten Teils von m_0 und eine Schranke T als Umfang der Offline-Arbeit, S und T haengen von der Schranke B(=2^b) ab.
- ***Input***: Size-Checking-Orakel mit oeffentl. Schluessel (e,N) und der Aufgabe $c_0 \equiv m_0^e \bmod N$, $0 < m_0 < B$, m_0 nur dem Orakel bekannt
- ***Output***: Wenn erfolgreich, dann die Nachricht m_0
- ***Variablen***: m', der aktuell bekannte glatte Teil von m_0; c' die RSA-Entschluesselung von m_0/m'

3.Angriff auf RSA (Teilbarkeitsangriff)

Step 1 Initialisiere $m' \leftarrow 1$, $c' \leftarrow c_0$

Step 2 FOR alle $p \leq S$, p prim (ggT(p,N)=1) DO

(a) Berechne die naechste Anfrage $c=c'/p^e \bmod N$

(b) IF ***O***(c) = ***success*** (dh p | m_0/m') THEN

setze $m' \leftarrow pm' \bmod N$, $c' \leftarrow c'/p^e \bmod N$ und GOTO Step 2(a)

(c) ELSE setze p auf die naechste Primzahl

Step 3 FOR alle $m_t \leq T$ DO

IF ***O***($c' \equiv m_t^e \bmod N$) = ***success*** THEN

gib $m \leftarrow m_t m' \bmod N$ als Kandidat von m_0 zurueck

Step 4 Abbruch, da m_0 nicht mit diesen Werten erhalten werden kann

Analyse des Teilbarkeits-Angriffs

- Die Korrektheit basiert auf der Invariante die man aus Step 1 und 2 erhaelt:

  $c \equiv (m')^e c' \bmod N$

  In Step 2 werden m' und c' genau dann veraendert, wenn die Propositionbesagt, dass die Entschluesselung von c' von p geteilt wird. In dieser Schleife wird der S-glatte Teil von m_0 extrahiert.

  Step 3 versucht dann den nicht-S-glatten-Teil der Nachricht zu finden.

- Angriff geht fuer Nachrichten m_0, die durch Primzahlen $p < S$ teilbar sind, bis auf evtl einen Faktor beschraenkt durch T. Das heisst die Erfolgs-Wahrscheinlichkeit ist durch die Wahrscheinlichkeit gegeben, dass eine zufaellige Nachricht von dieser Form ist.

Praktisches zum 3. Angriff

- Annahme: Schluessel fuer 2-key Triple-DES mit korrekten Pruef-Bits(dh Schluessel mit 128 Bit, aber nur 112 Entropie-Bits) werden mit reinem RSA und einem 1024 Bit Modulus verschluesselt.
- Die Wahrscheinlichkeit fuer semi-Glattheit liefert eine untere Schranke fuer die Erfolgswahrscheinlichkeit
- → $B=2^{128}$. Benutzt man $S=2^{22}$ und $T=2^{42}$, so ist die Schranke, gegeben durch die Wahrscheinlichkeit der semi-Glattheit fest.
- → Tests mit $22*2^{14}=360448$ zufaelligen 128 Bit-Zahlen liefern 559 Zahlen, die (T,S)-semi-glatt sind, heisst: $0{,}155\% \approx 2^{-9.3}$

Abschliessendes

- Integriaets-Check sollen zum richtigen Zeitpunkt geschehen (vorzugsweise direkt nach der Entschluesselung)

- Jeder Prozess, der nicht zw. Entschlueselung und Integritaets-Check liegen muss, sollte verschoben werden.

- Ein Server (das Orakel), der auf Fehler identisch reagiert, keinen Zugang zu Logfiles (von Personen, die nicht den privaten Schluessel besitzen), eine spezifische Anordnung von Unterroutinen und keine grossen Zeitunterschiede zwischen Teilfunktionen sind die Standarts, die erreicht werden sollten. Ob das je erreicht wird ist sehr unsicher.

Danke...

...fuer die Aufmerksamkeit.