Typical Flaws in Cryptography

Henning Seidler

November 11, 2019

What you should know for CTFs ...
please install IPython, optionally Sagemath

Cryptography

Alice wants to send message m to Bob, without Eve reading it.

encrypt: $c = \mathrm{enc}(m, k_1)$ (can be ambiguous)

decrypt: $m = \mathrm{dec}(c, k_2)$

Cryptography

Alice wants to send message m to Bob, without Eve reading it.

encrypt: $c = \mathrm{enc}(m, k_1)$ (can be ambiguous)

decrypt: $m = \mathrm{dec}(c, k_2)$

symmetric: $k_1 = k_2$

asymmetric: $k_1 \neq k_2$, but related

Cryptography

Alice wants to send message m to Bob, without Eve reading it.

encrypt: $c = \mathrm{enc}(m, k_1)$ (can be ambiguous)

decrypt: $m = \mathrm{dec}(c, k_2)$

symmetric: $k_1 = k_2$

asymmetric: $k_1 \neq k_2$, but related

Kerckhoff's Principle

$\mathrm{enc}(,)$ and $\mathrm{dec}(,)$ are known,
only keys k_1, k_2 secret

RSA

Setup:

Classic in Public Key Crypto

p, q primes

$n := p \cdot q \implies \varphi(n) = (p-1)(q-1)$

choose e coprime to $\varphi(n)$

$d := e^{-1} \bmod \varphi(n)$ (extended Euclidean Algorithm)

RSA

Setup:

Classic in Public Key Crypto

p, q primes

$n := p \cdot q \implies \varphi(n) = (p-1)(q-1)$

choose e coprime to $\varphi(n)$

$d := e^{-1} \bmod \varphi(n)$ (extended Euclidean Algorithm)

Keys:

public key (n, e)

private key (n, d), possibly $p, q, \varphi(n)$

RSA

Setup:

Classic in Public Key Crypto

p, q primes

$n := p \cdot q \implies \varphi(n) = (p-1)(q-1)$

choose e coprime to $\varphi(n)$

$d := e^{-1} \bmod \varphi(n)$ (extended Euclidean Algorithm)

Keys:

public key (n, e)

private key (n, d), possibly $p, q, \varphi(n)$

Usage:

encrypt $c = m^e \bmod n$

decrypt $m = c^d \bmod n$

RSA

Setup:

Classic in Public Key Crypto

p, q primes

$n := p \cdot q \implies \varphi(n) = (p-1)(q-1)$

choose e coprime to $\varphi(n)$

$d := e^{-1} \bmod \varphi(n)$ (extended Euclidean Algorithm)

Keys:

public key (n, e)

private key (n, d), possibly $p, q, \varphi(n)$

Usage:

encrypt $c = m^e \bmod n$

decrypt $m = c^d \bmod n$

Details: see Section 3 (Appendix)

Scenario

Hybrid Encryption: want to send AES key (128/256 Bit) via RSA (4096 Bit)

Scenario

Hybrid Encryption: want to send AES key (128/256 Bit) via RSA (4096 Bit)

Simple Case

$m < \sqrt[e]{n}$, also $m^e < n \implies m^e \bmod n = m^e$

don't compute modulo

Decrypt: $m = \sqrt[e]{c}$ in $\mathbb{Z}$ (bisection)

(!) standard code (float) fails for large numbers (!)

Scenario

Hybrid Encryption: want to send AES key (128/256 Bit) via RSA (4096 Bit)

Simple Case

$m < \sqrt[e]{n}$, also $m^e < n \implies m^e \bmod n = m^e$

don't compute modulo

Decrypt: $m = \sqrt[e]{c}$ in $\mathbb{Z}$ (bisection)

(!) standard code (float) fails for large numbers (!)

Example (hxp-CTF 2018)

$m > \sqrt[e]{n}$, but slightly;
$m^e = c + k \cdot n$ for small $k \in \mathbb{N}$
$\implies$ guess k then as above

Padding

Definition (Padding)

artificially enlarge message

Padding

Definition (Padding)

artificially enlarge message

Version 1: multiply with fixed number

Encrypt: $m' := m \cdot r$ for fixed $r \in \mathbb{Z}_n^*$; $c = (m')^e \mod n$

Padding

Definition (Padding)

artificially enlarge message

Version 1: multiply with fixed number

Encrypt: $m' := m \cdot r$ for fixed $r \in \mathbb{Z}_n^*$; $c = (m')^e \mod n$

Decrypt

Put $c' := (r^{-1})^e \cdot c$, then $c' = m^e$ (m small); decrypt as before

Padding

Definition (Padding)

artificially enlarge message

Version 1: multiply with fixed number

Encrypt: $m' := m \cdot r$ for fixed $r \in \mathbb{Z}_n^*$; $c = (m')^e \mod n$

Decrypt

Put $c' := (r^{-1})^e \cdot c$, then $c' = m^e$ (m small); decrypt as before

Version 2

concatenate m with itself: $m' = m||\ldots||m$

$\Longrightarrow$ $m = m \cdot 10\ldots010\ldots01\ldots01 \rightsquigarrow$ Version 1

Coppersmith

Theorem (Coppersmith)

Let $f \in \mathbb{Z}[t]$ normalised. Then we can efficiently compute $x \in \mathbb{Z}$ with $f(x) \equiv 0 \mod n$ and $|x| < \sqrt[\deg(f)]{n}$ (if it exists).

Coppersmith

Theorem (Coppersmith)

Let $f \in \mathbb{Z}[t]$ normalised. Then we can efficiently compute $x \in \mathbb{Z}$ with $f(x) \equiv 0 \mod n$ and $|x| < \sqrt[\deg(f)]{n}$ (if it exists).

Corollary

If m has fewer than $\log(n)/e$ bits, and we have fixed padding, we can compute m.

Coppersmith

Theorem (Coppersmith)

Let $f \in \mathbb{Z}[t]$ normalised. Then we can efficiently compute $x \in \mathbb{Z}$ with $f(x) \equiv 0 \mod n$ and $|x| < \sqrt[\deg(f)]{n}$ (if it exists).

Corollary

If m has fewer than $\log(n)/e$ bits, and we have fixed padding, we can compute m.

In Sagemath implemented as `f.small_roots()`.

Coppersmith

Theorem (Coppersmith)

Let $f \in \mathbb{Z}[t]$ normalised. Then we can efficiently compute $x \in \mathbb{Z}$ with $f(x) \equiv 0 \mod n$ and $|x| < \sqrt[\deg(f)]{n}$ (if it exists).

Corollary

If m has fewer than $\log(n)/e$ bits, and we have fixed padding, we can compute m.

In Sagemath implemented as `f.small_roots()`.

Example (PWN-CTF, Whistle)

Padding: $m' = 0001FF \ldots FF00||m$
$n \sim 4096$ bit, $e = 3$, $m \sim 128$ bit

Coppersmith – Exercise

Get you hands dirty

`http://page.math.tu-berlin.de/~seidler/crypto_class`

in `exercise_1` you have the source code

used key `pubkey.pem`

true message replaced by “?”

the 5 given cipher are the 5 lines of `output_1`

maybe don’t solve in given order

Håstad Broadcast

Szenario

Viele Empfänger mit kleinem e. Sende die selbe Nachricht an $\geq e$ Empfänger.

Håstad Broadcast

Szenario

Viele Empfänger mit kleinem e. Sende die selbe Nachricht an $\geq e$ Empfänger.

Example

Einladung zum CTF in der AGRS.

Håstad Broadcast

Szenario

Viele Empfänger mit kleinem e. Sende die selbe Nachricht an $\geq e$ Empfänger.

Example

Einladung zum CTF in der AGRS.

Chinesischer Restsatz

System an Kongruenzen mit teilerfremden n_i

$$c_i \equiv x \mod n_i \qquad\qquad i = 1, \ldots, e$$

Lösung ist $x = (m^e \bmod \prod n_i) = m^e$, dann einfach Wurzel ziehen.

Håstad – Übung

Hände schmutzig machen

`http://page.math.tu-berlin.de/~seidler/crypto_class`

in `exercise_2` ist der Quelltext

die 20 verwendeten Schlüssel sind in `keys`

echte Nachricht ersetzt durch "???"

die 20 Chiffren sind die 20 Dateien in `messages`

Kleine Nachricht m

Meet-in-the-Middle:

Sei $m = m_1 m_2$ in $\mathbb{Z}$, $m_i \leq 2^{b_i}$, $c = m^e \mod n$.

$\Longrightarrow$ $cm_1^{-e} \equiv m_2^e \mod n$.

- Liste $m_2^e \mod n$ für alle $m_2 \leq 2^{b_2}$
- Liste $cm_1^{-e} \mod n$ für alle $m_1 \leq 2^{b_1}$
- Suche Kollision

Kleine Nachricht m

Meet-in-the-Middle:
Sei $m = m_1 m_2$ in $\mathbb{Z}$, $m_i \leq 2^{b_i}$, $c = m^e \mod n$.
$\Longrightarrow$ $cm_1^{-e} \equiv m_2^e \mod n$.

- Liste $m_2^e \mod n$ für alle $m_2 \leq 2^{b_2}$
- Liste $cm_1^{-e} \mod n$ für alle $m_1 \leq 2^{b_1}$
- Suche Kollision

Zeit $\mathcal{O}(2^{b_1} + 2^{b_2})$ Speicher $\mathcal{O}\left(2^{\min(b_1,b_2)}\right)$

also etwa $\mathcal{O}\left(\sqrt{m}\right)$

Die Wahrscheinlichkeit, dass eine 64-Bit Zahl in zwei gleich große Teile zerfällt liegt bei 18%.

Kleine Nachricht – Übung

Hände schmutzig machen

`http://page.math.tu-berlin.de/~seidler/crypto_class`

in `exercise_3` ist der Quelltext

verwendeter Schlüssel `pubkey3.pem`

echte Nachrichten ersetzt durch "?"

die Chiffre steht in `output_3`

Tipp: $m_1 \leq 100.000$ (schont den RAM)

Kettenbrüche

Zwischenergebnisse von Euklid für Näherungsbrüche nutzen:

Input: $r_0 := \frac{a_0}{b_0}$

$r_i = \lfloor r_i \rfloor + \frac{1}{r_{i+1}}$ d.h. $\lfloor r_i \rfloor = (a_i // b_i)$ $r_{i+1} = (b_i \bmod a_i)/a_i$

Kettenbrüche

Zwischenergebnisse von Euklid für Näherungsbrüche nutzen:

Input: $r_0 := \frac{a_0}{b_0}$

$r_i = \lfloor r_i \rfloor + \frac{1}{r_{i+1}}$ d.h. $\lfloor r_i \rfloor = (a_i // b_i)$ $r_{i+1} = (b_i \bmod a_i)/a_i$

geht auch irrational: Beispiel goldener Schnitt

$$\varphi = 1 + \frac{1}{1 + \frac{1}{1 + \frac{1}{1 + \ldots}}} =: [1; 1, 1, 1, 1, \ldots]$$

Kettenbrüche

Zwischenergebnisse von Euklid für Näherungsbrüche nutzen:

Input: $r_0 := \frac{a_0}{b_0}$

$r_i = \lfloor r_i \rfloor + \frac{1}{r_{i+1}}$ d.h. $\lfloor r_i \rfloor = (a_i // b_i)$ $r_{i+1} = (b_i \bmod a_i)/a_i$

geht auch irrational: Beispiel goldener Schnitt

$$\varphi = 1 + \frac{1}{1 + \frac{1}{1 + \frac{1}{1 + \ldots}}} =: [1; 1, 1, 1, 1, \ldots]$$

i-te Näherung: den . . .-Term weglassen

Bruch ausrechnen

Eigenschaften von Kettenbrüchen

wird immer besser; gerade $\rightarrow$ kleiner; ungerade $\rightarrow$ größer

$$\frac{p_{2i}}{q_{2i}} < \frac{p_{2(i+1)}}{q_{2(i+1)}} \leq r_0 \leq \frac{p_{2(i+1)+1}}{q_{2(i+1)+1}} < \frac{p_{2i+1}}{q_{2i+1}}$$

Eigenschaften von Kettenbrüchen

wird immer besser; gerade $\rightarrow$ kleiner; ungerade $\rightarrow$ größer

$$\frac{p_{2i}}{q_{2i}} < \frac{p_{2(i+1)}}{q_{2(i+1)}} \leq r_0 \leq \frac{p_{2(i+1)+1}}{q_{2(i+1)+1}} < \frac{p_{2i+1}}{q_{2i+1}}$$

gute Näherung: $\left| r_0 - \frac{p_i}{q_i} \right| < \frac{1}{q_i^2}$

beste Näherung: Sei $\frac{p_i}{q_i}$ ein Kettenbruch von r_0.

Wenn $\frac{p}{q}$ bessere Näherung, dann $q > q_i$.

Eigenschaften von Kettenbrüchen

wird immer besser; gerade $\rightarrow$ kleiner; ungerade $\rightarrow$ größer

$$\frac{p_{2i}}{q_{2i}} < \frac{p_{2(i+1)}}{q_{2(i+1)}} \leq r_0 \leq \frac{p_{2(i+1)+1}}{q_{2(i+1)+1}} < \frac{p_{2i+1}}{q_{2i+1}}$$

gute Näherung: $\left| r_0 - \frac{p_i}{q_i} \right| < \frac{1}{q_i^2}$

beste Näherung: Sei $\frac{p_i}{q_i}$ ein Kettenbruch von r_0.

Wenn $\frac{p}{q}$ bessere Näherung, dann $q > q_i$.

"'einzige"' Näherung: $\left| r_0 - \frac{p}{q} \right| < \frac{1}{2q^2} \implies \frac{p}{q}$ ist Kettenbruch

Kleiner privater Schlüssel d

Theorem (Wiener, 1990)

Sei $q < p < 2q$ und $d < \frac{1}{3}n^{\frac{1}{4}}$. Dann kann der private Schlüssel d aus (n, e) in $\mathcal{O}(\log(n)^2)$ berechnet werden.

Kleiner privater Schlüssel d

Theorem (Wiener, 1990)

Sei $q < p < 2q$ und $d < \frac{1}{3}n^{\frac{1}{4}}$. Dann kann der private Schlüssel d aus (n, e) in $\mathcal{O}(\log(n)^2)$ berechnet werden.

Vorgehen: Approximiere $\frac{e}{n}$ mit Kettenbrüchen.

Schreibe $ed - k\varphi(n) = 1$

Liefert Abschätzung $\left|\frac{e}{n} - \frac{k}{d}\right| < \ldots < \frac{1}{2d^2}$

Kleiner privater Schlüssel d

Theorem (Wiener, 1990)

Sei $q < p < 2q$ und $d < \frac{1}{3}n^{\frac{1}{4}}$. Dann kann der private Schlüssel d aus (n, e) in $\mathcal{O}(\log(n)^2)$ berechnet werden.

Vorgehen: Approximiere $\frac{e}{n}$ mit Kettenbrüchen.

Schreibe $ed - k\varphi(n) = 1$

Liefert Abschätzung $\left|\frac{e}{n} - \frac{k}{d}\right| < \ldots < \frac{1}{2d^2}$

$\Longrightarrow$ $\frac{k}{d}$ ist ein Kettenbruch von $\frac{e}{n}$

Kleiner privater Schlüssel d

Theorem (Wiener, 1990)

Sei $q < p < 2q$ und $d < \frac{1}{3}n^{\frac{1}{4}}$. Dann kann der private Schlüssel d aus (n, e) in $\mathcal{O}(\log(n)^2)$ berechnet werden.

Vorgehen: Approximiere $\frac{e}{n}$ mit Kettenbrüchen.

Schreibe $ed - k\varphi(n) = 1$

Liefert Abschätzung $\left|\frac{e}{n} - \frac{k}{d}\right| < \ldots < \frac{1}{2d^2}$

$\Longrightarrow$ $\frac{k}{d}$ ist ein Kettenbruch von $\frac{e}{n}$

Kettenbrüche $\rightsquigarrow$ Liste von Kandidaten

a) Test, ob Entschlüsselung sinnvoll

b) Versuche n damit zu faktorisieren

Kleines d – Übung

Hände schmutzig machen

`http://page.math.tu-berlin.de/~seidler/crypto_class`

in `exercise_4` ist der Quelltext

echte Nachrichten ersetzt durch "?"

2 Beispiele: 1024 Bit und 4096 Bit

Ein Riss macht alles kaputt

Theorem (Theorem der geheimen Parameter)

Aus einem Teil des geheimen Schlüssels $(p, q, \varphi(n), d)$ und dem öffentlichen Schlüssel können die anderen Teile des geheimen Schlüssels effizient berechnet werden.

Ein Riss macht alles kaputt

Theorem (Theorem der geheimen Parameter)

Aus einem Teil des geheimen Schlüssels $(p, q, \varphi(n), d)$ und dem öffentlichen Schlüssel können die anderen Teile des geheimen Schlüssels effizient berechnet werden.

Corollary

Wenn d bekannt wurde, reicht es nicht, einfach e und d auszutauschen.

Ein Riss macht alles kaputt

Theorem (Theorem der geheimen Parameter)

Aus einem Teil des geheimen Schlüssels $(p, q, \varphi(n), d)$ und dem öffentlichen Schlüssel können die anderen Teile des geheimen Schlüssels effizient berechnet werden.

p, q **bekannt** berechne alles, wie bei Erstellung des Schlüssels

$\varphi(n)$ **bekannt** quadratische Gleichung lösen:

$$a := n - \varphi(n) = p + q - 1 \qquad \text{bekannt}$$
$$n = p \cdot q = p \cdot (a + 1 - p)$$

Löse nach p, teste beide Lösungen, ob korrekt

$\Longrightarrow$ Faktoren p, q

Ein Riss macht alles kaputt

Theorem (Theorem der geheimen Parameter)

Aus einem Teil des geheimen Schlüssels $(p, q, \varphi(n), d)$ und dem öffentlichen Schlüssel können die anderen Teile des geheimen Schlüssels effizient berechnet werden.

Sei also d bekannt.

e **klein** $ed - 1 = x \cdot \varphi(n)$ in $\mathcal{O}(e)$ ausprobieren

sonst Wähle a mit $a \not\equiv 0 \mod n$ zufällig. $s := V_2(ed-1)$, $k = \frac{ed-1}{2^s}$

Teste $\mathrm{ggT}(a, n) \neq 1$

Teste $\mathrm{ggT}((a^k)^{2^i} - 1, n) \neq 1$ für $0 \leq i \leq s-1$

Wenn dies $\neq 1$,dann p oder q gefunden.

Erfolgswahrscheinlichkeit $\geq \frac{1}{2}$

Schlüssel zerlegen – Übung

Hände schmutzig machen

`http://page.math.tu-berlin.de/~seidler/crypto_class`

in `exercise_5` ist der Quelltext

echte Nachrichten ersetzt durch "?"

`key1` ist zu (indirekt) gegebenem p

`key2` und `key3` zu gegebenem d

Aufgabe 5.4 ist nur Schlüssel und $\varphi(n)$ gegeben, keine Nachricht; also einfach alle anderen Teile bestimmen

Primzahlen nah beieinander

Theorem

Wenn p, q nah beieinander liegen, können wir n faktorisieren.

Primzahlen nah beieinander

Theorem

Wenn p, q nah beieinander liegen, können wir n faktorisieren.

Wähle m mit $(m-1)^2 \leq n < m^2$

setze $\Delta_i = \sqrt{(m+i)^2 - n}$ für $i \in \mathbb{N}$

falls $\Delta_i \in \mathbb{N}$, setze $p = m + i - \Delta_i$

Laufzeit: $\mathcal{O}\left(\frac{p-q}{2\sqrt{2p}}\right)$ Durchläufe

Primzahlen nah beieinander

Theorem

Wenn p, q nah beieinander liegen, können wir n faktorisieren.

Wähle m mit $(m-1)^2 \leq n < m^2$

setze $\Delta_i = \sqrt{(m+i)^2 - n}$ für $i \in \mathbb{N}$

falls $\Delta_i \in \mathbb{N}$, setze $p = m + i - \Delta_i$

Laufzeit: $\mathcal{O}\left(\frac{p-q}{2\sqrt{2p}}\right)$ Durchläufe

Schreibe $n = p \cdot q = \left(\frac{p+q}{2}\right)^2 - \left(\frac{p-q}{2}\right)^2$

Am Ende gilt $\frac{p-q}{2} = \Delta_i$, zudem $m \approx p$, das ergibt $i \approx \frac{p-q}{2\sqrt{2p}}$

Primzahlen nah beieinander – Übung

Hände schmutzig machen

`http://page.math.tu-berlin.de/~seidler/crypto_class`

in `exercise_6` ist der Quelltext

erzeugt zufällig eine Zahl n, die ihr faktorisieren sollt

Gemeinsamer Modulus

Theorem

Wenn wir Schlüssel (n, e_1) *und* (n, e_2) *haben, mit* $\mathrm{ggT}(e_1, e_2) = 1$, *dann können wir alle Nachrichten lesen.*

Corollary

Jeder braucht eigenen Modulus n, also eigenen Primzahlen.

Gemeinsamer Modulus

Theorem

Wenn wir Schlüssel (n, e_1) *und* (n, e_2) *haben, mit* $\mathrm{ggT}(e_1, e_2) = 1$, *dann können wir alle Nachrichten lesen.*

Corollary

Jeder braucht eigenen Modulus n, also eigenen Primzahlen.

kennen Chiffres $c_i = m^{e_i} \bmod n$

Erweiterter Euklid: $se_1 + te_2 = 1$, mit $s < 0$ und $t > 0$

berechne $c_1^{-1} \bmod n$ (wenn Fehlschlag, dann Faktor p von n)

$$\left(c_1^{-1}\right)^{|s|} c_2^t \equiv \left(\left(m^{e_1}\right)^{-1}\right)^{|s|} \left(m^{e_2}\right)^t \equiv m^{se_1+te_2} \equiv m \mod n$$

haben Nachricht m berechnet

Gemeinsamer Modulus – Übung

Hände schmutzig machen

`http://page.math.tu-berlin.de/~seidler/crypto_class`

in `exercise_7` ist der Quelltext

echte Nachrichten ersetzt durch “?”

(Kryptographische) Hash-Funktionen

Funktion $h : \{0,1\}^* \to \{0,1\}^n$

einfach zu berechnen

Urbild, Kollision schwer zu berechnen

(Kryptographische) Hash-Funktionen

Funktion $h : \{0,1\}^* \to \{0,1\}^n$

einfach zu berechnen

Urbild, Kollision schwer zu berechnen

Lawineneffekt: Bitflip in Eingabe soll alle Bit mit Wahrscheinlichkeit $\frac{1}{2}$ flippen

Funktionswerte etwa gleich verteilt

(Kryptographische) Hash-Funktionen

Funktion $h : \{0,1\}^* \to \{0,1\}^n$

einfach zu berechnen

Urbild, Kollision schwer zu berechnen

Lawineneffekt: Bitflip in Eingabe soll alle Bit mit Wahrscheinlichkeit $\frac{1}{2}$ flippen

Funktionswerte etwa gleich verteilt

Theorem (Geburtstagsparadox)

Wenn ein Jahr n Tage hat, dann haben aus $\frac{6}{5}\sqrt{n}$ Menschen mit Wahrscheinlichkeit $\geq \frac{1}{2}$ zwei am selben Tag Geburtstag.

Hash-Kollisionen

Theorem (Floyds Algorithmus)

Sei $h : X \to Z \subseteq X$ eine Hashfunktion, $x_0 \in X \setminus Z$. Dann können wir eine Kollision in Zeit $\mathcal{O}(\sqrt{n})$ und Platz $\mathcal{O}(1)$ berechnen.

Hash-Kollisionen

Theorem (Floyds Algorithmus)

Sei $h : X \to Z \subseteq X$ eine Hashfunktion, $x_0 \in X \setminus Z$. Dann können wir eine Kollision in Zeit $\mathcal{O}(\sqrt{n})$ und Platz $\mathcal{O}(1)$ berechnen.

Abkürzung: $x_{i+1} := h(x_i)$

$a := x_1,\ b := x_2,\ \delta = 1$
while $a \neq b$ **do**
    $a := h(a),\ b := h(h(b)),\ \delta = \delta + 1$
end while
$a = x_1,\ b = x_{\delta+1}$
while $h(a) \neq h(b)$ **do**
    $a := h(a),\ b = h(b)$
end while
return (a, b)

Urbild – Meet in the Middle

$$h(c_1 \dots c_n) = [s_0 = \mathsf{IV}; s_{i+1} = f(s_i, c_{i+1}); \texttt{return } s_n]$$

Theorem

Wenn man f invertieren kann, kann man ein Urbild aus 2^{2n} Kandidaten mit $\mathcal{O}(2^n)$ Speicher und $\mathcal{O}(n \cdot 2^n)$ Zeit durchsuchen.

Urbild – Meet in the Middle

$$h(c_1 \dots c_n) = [s_0 = \mathrm{IV}; s_{i+1} = f(s_i, c_{i+1}); \texttt{return } s_n]$$

Theorem

Wenn man f invertieren kann, kann man ein Urbild aus 2^{2n} Kandidaten mit $\mathcal{O}(2^n)$ Speicher und $\mathcal{O}(n \cdot 2^n)$ Zeit durchsuchen.

Input: $h(x_1 \dots x_{2n})$ mit unbekanntem x

speichere alle $h(c_1 \dots c_n)$, Liste sortieren

"'Umkehrung"': $h^{-1} : h(x) \mapsto \mathrm{IV}$, nach obigem Schema

für alle $h^{-1}(c_1 \dots c_n)$ teste, ob in Liste

Urbild – Meet in the Middle

$$h(c_1 \dots c_n) = [s_0 = \mathrm{IV}; s_{i+1} = f(s_i, c_{i+1}); \texttt{return } s_n]$$

Theorem

Wenn man f invertieren kann, kann man ein Urbild aus 2^{2n} Kandidaten mit $\mathcal{O}(2^n)$ Speicher und $\mathcal{O}(n \cdot 2^n)$ Zeit durchsuchen.

Input: $h(x_1 \dots x_{2n})$ mit unbekanntem x

speichere alle $h(c_1 \dots c_n)$, Liste sortieren

"'Umkehrung"': $h^{-1} : h(x) \mapsto \mathrm{IV}$, nach obigem Schema

für alle $h^{-1}(c_1 \dots c_n)$ teste, ob in Liste

erlaubt 56 Bit Brute-Force mit Laptop in wenigen Minuten

Hashing – Übung

Hände schmutzig machen

`http://page.math.tu-berlin.de/~seidler/crypto_class`

in `exercise_8` ist der Quelltext

1. finde Kollision von h
2. finde Urbild x zu gegebenem $h(x)$

Signaturen

Ziel:

sicher stellen, dass Nachricht wirklich vom Absender kommt

keine Manipulation

Signaturen

Ziel:

sicher stellen, dass Nachricht wirklich vom Absender kommt

keine Manipulation

Möglichkeit:

mit privatem Schlüssel ”‘entschlüsseln”’

”‘Verschlüsseln”’ mit öffentlichem Schlüssel liefert Original

Signaturen

Ziel:

sicher stellen, dass Nachricht wirklich vom Absender kommt

keine Manipulation

Möglichkeit:

mit privatem Schlüssel "'entschlüsseln"'

"'Verschlüsseln"' mit öffentlichem Schlüssel liefert Original

Problem: manipulierte Nachricht könnte valider Code sein (z.B. bei RSA ist jede Zahl zulässiger Code)

Lösung: Sende $(m, \text{sign}(\text{hash}(m)))$

Signaturangriff auf RSA

Problem bei RSA: Signieren ist multiplikativ:

$$m_1^d \cdot m_2^d \equiv (m_1 m_2)^d \mod n$$

Wenn wir Signatur von m_1, m_2 haben, dann auch von $m := m_1 \cdot m_2$

Signaturangriff auf RSA

Problem bei RSA: Signieren ist multiplikativ:

$$m_1^d \cdot m_2^d \equiv (m_1 m_2)^d \mod n$$

Wenn wir Signatur von m_1, m_2 haben, dann auch von $m := m_1 \cdot m_2$

Angriff mit Orakel:

Ziel: Signiere m

Können manche Nachrichten signieren (per Orakel)

Signiere geeignete Faktoren von m

Signatur – Übung

Hände schmutzig machen

`http://page.math.tu-berlin.de/~seidler/crypto_class`

in `exercise_9` ist der Quelltext

Öffnet nur `cookie_sign` und den public-key.

Auf keinen Fall(!) `dont_look` oder den private-key öffnen.

Kongruenzen

Notation Kongruenz

$a \equiv b \mod n$ heißt einfach $(a \bmod n) = (b \bmod n)$

Theorem (Kleiner Satz von Fermat)

Sei p prim, dann gilt $a^p \equiv a \mod p$.
Alternativ: Sei p prim und a zu p teilerfremd ($\mathrm{ggT}(a, p) = 1$, bzw. hier: a kein Vielfaches von p). Dann gilt $a^{p-1} \equiv 1 \mod p$.

Theorem (Satz von Euler)

Seien $a, n \in \mathbb{Z}$ mit $\mathrm{ggT}(a, n) = 1$. Dann $a^{\varphi(n)} \equiv 1 \mod n$.

Erweiterter Euklidischer Algorithmus

Interface

Input $a, b \in \mathbb{N}$

Aufruf: $(d, x, y) = \text{ErwEuklid}(a, b)$

Output: $d \in \mathbb{N}$, $x, y \in \mathbb{Z}$ mit $d = \text{ggT}(a, b) = ax + by$

```
def EEA(a,b):
  if b = 0: return (a,1,0)
  d,s,t = EEA(b, a \% b)
  return (d, s, s - (a//b) * t)
```

Modulare Inverse

Anwendung RSA

ErwEuklid$(e, \varphi(n)) = (1, d, *)$

ggT ist immer 1 (teilerfremd)

letzten Wert $*$ brauchen wir nicht

Allgemein invertieren: $a^{-1} \bmod n$

$$\text{ErwEuklid}(a, n) = (1, a^{-1} \bmod n, *)$$

Chinesischer Restsatz

Theorem (Chinesischer Restsatz)

Seien $n_i \in \mathbb{Z}$ (paarweise) teilerfremd und $a_i \in \mathbb{Z}$ beliebig für $i = 1, \ldots, k$. Dann hat das System von Kongruenzen

$$a_i \equiv x \mod n_i \qquad i = 1, \ldots, k$$

eine eindeutige Lösung $0 \leq x < \prod n_i$.

Algorithmus für 2 Kongruenzen:

$$\begin{aligned} \text{ErwEuklid}(n_1, n_2) \rightsquigarrow & \quad 1 = s \cdot n_1 + t \cdot n_2 \\ \text{Lösung} & \quad x := a_2 \cdot s \cdot n_1 + a_1 \cdot t \cdot n_2 \end{aligned}$$

Rekursiv weiter: $c' = x$ und $n' = n_1 \cdot n_2$

RSA – typische Parameter

Definition (Fermat-Primzahl)

Eine Fermat-Primzahl ist eine Primzahl der Form $2^k + 1$.

Die einzigen bekannten sind $3, 5, 17, 257, 65537$.
Wähle man $e = 2^k + 1$, braucht man nur $k + 1 \leq 17$ Multiplikationen beim Verschlüsseln.
Beliebiges e braucht $\sim \frac{3}{2} \log n \approx 6000$ Multiplikationen.

d muss immer groß sein (siehe Wieners Angriff)

$\Longrightarrow$ Entschlüsselung braucht immer $\sim \frac{3}{2} \log n$ Multiplikationen