

Grundlagenpraktikum Netz- und Datensicherheit

Thema: **Angriffe auf RSA**

Lehrstuhl für Netz- und Datensicherheit
Ruhr-Universität Bochum

Versuchsdurchführung: Raum ID 2/168



Betreuung: Dominik Noß
Zusammengestellt von: Andreas Becker, Hannes Oberender, Florian Bache, Patrick Meier, Endres Puschner
Stand: 15. März 2020
Version: 2.0

Inhaltsverzeichnis

1	Vorwort	2
2	Grundlagen	3
2.1	Einleitung	3
2.2	RSA - Das Verfahren	3
2.3	Chinesischer Restsatz (CRT)	4
2.3.1	Lösung von simultanen Kongruenzen mit dem CRT	4
2.3.2	Berechnung von RSA-Signaturen mit dem CRT	5
3	Versuch	6
3.1	Voraussetzungen für die Teilnahme	6
3.2	Versuchsaufbau	6
3.3	Durchführung	6
3.3.1	RSA-Verschlüsselung mit kleinem öffentlichen Exponenten	6
3.3.2	Fault-Angriff gegen RSA-CRT	7
3.3.3	Bug-Attack	9
3.4	Kontrollfragen	12

1 Vorwort

Angriffe gegen kryptografische Systeme lassen sich, unabhängig vom konkreten System, wie folgt klassifizieren:

Brute Force-Angriffe gegen RSA sind aufgrund der großen, angepassten Schlüssellänge weitgehend ausgeschlossen. Jedoch gibt es eine Reihe von Angriffen, die sich gegen das Verfahren selbst richten. Dem gegenüber stehen die Implementierungsangriffe, die sich gezielt Fehler in konkreten Implementierungen des Verfahrens zunutze machen.

Die übrigen Klassen von Angriffen sollen in Form eines Versuches im Rahmen des Grundlagenpraktikums für IT-Sicherheit näher beleuchtet und praktisch durchgeführt werden. Dabei werden eigens hierfür erstellte Programme als System betrachtet, von denen lediglich bekannt ist, dass sie anfällig gegen bestimmte Angriffe sind.

Ziel ist es, einen Eindruck davon zu bekommen, dass kryptografische Verfahren keinesfalls im luftleeren Raum zu betrachten sind, sondern dass vielmehr der sichere Einsatz einer Chiffre von einer Reihe von Faktoren abhängt, während dem Angreifer die Ausnutzung von nur einer einzigen Sicherheitslücke genügt.

All dies soll am Fallbeispiel von RSA, der wohl populärsten asymmetrischen Chiffre, nachvollzogen werden.

Viel Spaß :-)

2 Grundlagen

2.1 Einleitung

RSA (Rivest, Shamir und Adleman, [?]) ist der Name eines 1977 veröffentlichten public key Verschlüsselungs- und Signaturverfahrens.

Die Sicherheit von RSA basiert auf dem Problem, große Zahlen zu faktorisieren. Wenn man große Zahlen in Polynomialzeit faktorisieren kann, kann man RSA brechen. Bisher wurde allerdings nur in eingeschränkten Berechnungsmodellen (Generic Ring Algorithms) gezeigt, dass RSA und Faktorisieren äquivalent sind.

2.2 RSA - Das Verfahren

Setup:

- wähle zwei zufällige große Primzahlen p, q
- berechne $n = p \cdot q$
- wähle eine Zahl e , sodass gilt:
 - $1 < e < \varphi(n)$, $\varphi(n) = (p-1) \cdot (q-1)$
 - $\text{ggT}(e, \varphi(n)) = 1$
- berechne $d = e^{-1} \bmod \varphi(n)$

$\Rightarrow \text{pk} := (n, e), \text{sk} := d$

Verschlüsselung:

- gegeben Klartext $m \in \mathbb{Z}_n$
- berechne $c = m^e \bmod n$

Entschlüsselung:

- gegeben Chiffre $c \in \mathbb{Z}_n$
- berechne $m = c^d \bmod n$

Signatur:

- gegeben Nachricht $m \in \mathbb{Z}_n$
- berechne $s = m^d \bmod n$

Verifikation:

- gegeben (m, s)
- prüfe, ob $m \stackrel{?}{=} s^e \bmod n$

RSA ist zugleich Public Key-Verschlüsselungssystem und Signaturverfahren. Das ist möglich, weil für alle RSA-Klartexte gilt: $m = \text{enc}_e(\text{dec}_d(m)) = \text{dec}_d(\text{enc}_e(m))$, das heißt Verschlüsselung und Entschlüsselung sind vertauschbar!

Praktisch vorkommende Nachrichten liegen nun in der Regel allerdings nicht in \mathbb{Z}_n . Daher wird zur Berechnung einer digitalen Signatur an Stelle von m oft das Bild einer Hashfunktion $x = h(m)$ gewählt, welches wieder in \mathbb{Z}_n liegt. Zur Verifikation muss dann überprüft werden, ob $h(m) \stackrel{?}{=} s^e \bmod n$ gilt. Beim Einsatz von Hashfunktionen werden weitere kryptographische Eigenschaften gefordert, wie die Einweg-Eigenschaft (zu einem gegebenen Bild kann das Urbild nicht berechnet werden) und die Kollisionsresistenz (es ist schwierig, zwei Urbilder zu finden, die auf das gleiche Bild abbilden). Ein Angreifer mit begrenzter Rechenleistung und begrenztem Speicherplatz kann diese Eigenschaften also nicht brechen.

Da RSA, verglichen mit symmetrischen Verschlüsselungsverfahren, extrem rechenintensiv ist, findet es in der Praxis selten Einsatz als pure Verschlüsselung. Stattdessen werden Hybrid-Verfahren benutzt,

in welchen RSA lediglich für den Schlüsseltransport benutzt wird. Dabei verschlüsselt man mit RSA einen Schlüssel für ein symmetrisches Verfahren, wie etwa AES. Mit diesem Schlüssel wird die geheime Nachricht verschlüsselt. Dieser verschlüsselte Schlüssel wird mitsamt der chiffrierten Nachricht übertragen.

Der Empfänger entschlüsselt dann mit seinem privaten Schlüssel den symmetrischen, mit welchem er dann die geheime Nachricht lesen kann.

Der Vorteil davon ist, dass die teure RSA-Ver- und Entschlüsselung nur jeweils ein einziges Mal für den symmetrischen Schlüssel berechnet werden muss. Die verhältnismäßig lange, geheime Nachricht wird mit AES viel schneller ver- und entschlüsselt.

Auf diese Aspekte wird im Rahmen dieses Praktikums nicht weiter eingegangen; Klartextnachrichten sind bewusst einfach gewählt, da der Fokus auf Angriffen gegen das RSA-Verfahren liegt. Ein letztes Wort noch zu praktischen Implementierungen: Um mit RSA signieren und verschlüsseln zu können, sind heutzutage Schlüssellängen und RSA-Module >1024 Bit vonnöten. Daher reichen bei der Implementierung einfache Datentypen gängiger Programmiersprachen nicht aus und man bedient sich in der Regel einer Bibliothek für Langzahlarithmetik, z.B. der GNU Multiprecision Arithmetic Library (GMP, [?]).

2.3 Chinesischer Restsatz (CRT)

Ein großes Manko von Public Key-Verfahren ist die rechenaufwändige Langzahlarithmetik. Während der öffentliche Schlüssel bei RSA (in der Praxis oft $e \in \{3, 2^{16} + 1, \dots\}$) verhältnismäßig kurz und Exponentiationen effizient berechenbar sind, hat der geheime Exponent d die volle Länge des Modulus n , also in der Praxis etwa zwischen 1024 und 4096 Bit.

Um die notwendigen Exponentiationen auch in Umgebungen mit begrenzten Ressourcen möglichst effizient durchführen zu können, bedient man sich in einigen Implementierungen des Chinesischen Restsatzes (CRT, [?]):

CRT 1 (Chinesischer Restsatz) Seien $p, q \in \mathbb{Z}$ mit $\text{ggT}(p, q) = 1$. Dann gilt: $\mathbb{Z}_{pq} \simeq \mathbb{Z}_p \times \mathbb{Z}_q$

d.h. die Ringe \mathbb{Z}_{pq} und $\mathbb{Z}_p \times \mathbb{Z}_q$ sind isomorph. Wir können also, statt in \mathbb{Z}_n zu rechnen, die Operanden vorher in die Ringe \mathbb{Z}_p und \mathbb{Z}_q abbilden (s.u.) und die Berechnungen zunächst in diesen Ringen durchführen, was wesentlich schneller geht, als in \mathbb{Z}_n . Anschließend müssen die Teilergebnisse wieder zusammengesetzt werden.

2.3.1 Lösung von simultanen Kongruenzen mit dem CRT

Der chinesische Restsatz beschreibt in der Form, wie er oben angegeben ist, die Existenz einer bijektiven Abbildung zwischen algebraischen Strukturen. Uns interessiert aber die konkrete Berechnung einer solchen Abbildung für ein System aus r simultanen Kongruenzen mit paarweise teilerfremden Modulen n_i (mit dem Ziel, modulare Exponentiationen zu beschleunigen, wie sie beim RSA vorkommen):

$$\begin{aligned} s &\equiv s_1 \pmod{n_1} \\ s &\equiv s_2 \pmod{n_2} \\ &\vdots \\ s &\equiv s_r \pmod{n_r} \end{aligned}$$

Ein s , welches diese Kongruenzen erfüllt, findet man wie folgt:

1. berechne $N = \prod_{i=1}^r n_i$
2. berechne $N_i = N/n_i \forall i \in \{1, \dots, r\}$
3. berechne die Inversen $N_i^{-1} \bmod n_i \forall i \in \{1, \dots, r\}$ mittels EEA
4. berechne die Lösung wie folgt:

$$s = \left(\sum_{i=1}^r s_i \cdot (N_i^{-1} \bmod n_i) \cdot N_i \right) \bmod N$$

2.3.2 Berechnung von RSA-Signaturen mit dem CRT

Hier wird im Folgenden beschrieben, wie eine RSA-Signatur mit Hilfe des chinesischen Restsatzes (RSA-CRT) berechnet werden kann, was dem einfachen Fall von zwei simultanen Kongruenzen, $s \bmod p$ und $s \bmod q$, entspricht.

Gegeben seien dazu der geheime Schlüssel d , sowie die Faktorisierung des öffentlichen Moduls n , p und q . Dann kann die Berechnung der Signatur $s = m^d \bmod n$ wie folgt aufgeteilt werden:

1. Aufteilen der Nachricht m in m_p und m_q :

$$\begin{aligned} m_p &= m \bmod p \\ m_q &= m \bmod q \end{aligned}$$

2. Aufteilen des geheimen Schlüssels d in d_p und d_q :

$$\begin{aligned} d_p &= d \bmod \varphi(p) \\ d_q &= d \bmod \varphi(q) \end{aligned}$$

3. Berechnen der Teilsignaturen s_p und s_q :

$$\begin{aligned} s_p &= m_p^{d_p} \bmod p \\ s_q &= m_q^{d_q} \bmod q \end{aligned}$$

4. Nun gilt für die Gesamtsignatur:

$$\begin{aligned} s &\equiv s_p \bmod p \\ s &\equiv s_q \bmod q \end{aligned}$$

Für diesen einfachen Fall von zwei simultanen Kongruenzen kann die Gesamtsignatur s wie folgt mit Hilfe des CRT berechnet werden, unter Zuhilfenahme des erweiterten euklidischen Algorithmus zum Berechnen der multiplikativen Inversen $p^{-1} \bmod q$ und $q^{-1} \bmod p$:

$$s = (q^{-1} \bmod p) \cdot m_p^{d_p} \cdot q + (p^{-1} \bmod q) \cdot m_q^{d_q} \cdot p$$

3 Versuch

3.1 Voraussetzungen für die Teilnahme

- Dieses Dokument muss vorher gelesen und verstanden worden sein
- Sie sollten ein grundlegendes Verständnis von Public Key-Systemen im Allgemeinen und RSA im Speziellen haben
- Sie sollten die multiplikative Inverse einer Zahl in \mathbb{Z}_n mit Hilfe des EEA berechnen können
- Sie sollten grundlegende Erfahrung im Umgang mit Linux haben, siehe Linux-Befehlsreferenz!
- Sie müssen in der Lage sein, die Kontrollfragen zu beantworten.

3.2 Versuchsaufbau

Die für diesen Versuch speziell angefertigten RSA-Implementierungen stehen Ihnen als Webapplikationen zur Verfügung.

Zur Durchführung von Berechnungen mit großen Zahlen können Sie Python nutzen. Mit dem Befehl `python` starten Sie eine interaktive Pythonumgebung. Dort sind auch Variablenzuweisung möglich, die Ihnen erlauben mehrere Werte zu speichern und diese in weiteren Rechnungen beliebig zu nutzen.

Wichtig: Sinn und Zweck des Versuchs ist es, Ihnen ein Gespür dafür zu vermitteln, welche Schwachstellen in den Programmen vorhanden sind und wie ein konkreter Angriff dagegen aussieht. Sie dürfen einen Taschenrechner bzw. Webseiten mit ggT-Funktion benutzen, um Berechnungen durchzuführen.

Unerwünscht sind hingegen Lösungen mittels Brute Force oder Reverse Engineering!

3.3 Durchführung

3.3.1 RSA-Verschlüsselung mit kleinem öffentlichen Exponenten

Sie haben in Erfahrung gebracht, dass drei der schwergewichtigsten Mafiosi der heutigen Zeit eine RSA-verschlüsselte Nachricht zugestellt bekommen haben. Ein Praktikant hat für Sie bereits PEM¹-Dateien mit den öffentlichen Schlüsseln der Ganoven besorgt.

Schritt 1: Ermittlung der RSA-Parameter Öffnen Sie ein Konsolenfenster und begeben Sie sich in das Verzeichnis `/home/student/Praktikum/rsa`. Dort finden Sie Dateien mit den base64-kodierten öffentlichen Schlüsseln `costa.pk`, `gambino.pk` und `zarella.pk`.

Lassen Sie sich die öffentlichen RSA-Parameter auf der Konsole ausgeben!

Wie lautet der vollständige OpenSSL-Befehl `[?]` mit den notwendigen Parametern?

Welche öffentlichen Schlüssel e und welche Module n benutzen die drei Mafiosi?

Schritt 2: Bestimmen der simultanen Kongruenz Ihr Ziel ist es, eine Ihnen unbekannte Nachricht m in Erfahrung zu bringen, von der sie vermuten, dass sie an drei verschiedene Empfänger mittels RSA verschlüsselt wurde. Die Reihenfolge der Empfänger ist durch die obige Aufzählung gegeben.

¹Datenformat für Base64-kodierte Daten, die von Anfang- und Ende-Tags eingeschlossen sind.

Die Chifftrate der Nachrichten, die an die drei Mafiosi verschlüsselt wurden, sind:

$$c_1 = 1786058302$$

$$c_2 = 521332323$$

$$c_3 = 2806221682$$

Berechnen Sie daraus den zugehörigen Klartext m , indem sie die ein Gleichungssystem aufstellen und die simultanen Kongruenzen lösen [?]. Um multiplikative Inverse mit dem EEA zu berechnen, sind z.B. die Mathematikseiten von Arndt Brünnner hilfreich: <http://www.arndt-bruenner.de/mathe/mathekurse.htm>. Die notwendigen Rechenschritte müssen klar erkennbar sein!

Tip: Die N_i zuerst modulo n_i reduzieren!

Schritt 3: Dekodieren des Klartextes Interpretieren Sie das Ergebnis als Folge von ASCII-Zeichen in Dezimaldarstellung. Jedes Zeichen wird als drei dezimale Ziffern dargestellt, mit Ausnahme einer führenden Null. Welche streng geheime Nachricht wurde den Mafiosi zugesandt?

3.3.2 Fault-Angriff gegen RSA-CRT

Bei der Fault-Attack induziert ein Angreifer während einer Berechnung (beispielsweise auf einer Smartcard) mit einem kryptografischen Geheimnis gezielt Fehler, um Rückschlüsse auf das Geheimnis ziehen zu können.

Besuchen sie mithilfe eines Webbrowsers die lokale URL <http://192.168.1.6/smartcard>. Mit der dort bereitgestellten Webapplikation haben Sie die Möglichkeit, eine simulierte Smartcard Nachrichten (ganzahlige Werte) signieren zu lassen:



Abbildung 1: Webapplikation smartcard

Sie wissen außerdem, dass der verbaute Chip anfällig ist gegen verschiedene Faultangriffe. Vorab beantworten Sie in ihrer Ausarbeitung bitte folgende Fragen, die der Motivation und der Hinführung dienen:

Schritt 1: Mathematische Vorarbeit

- Wie wird die RSA-Signatur als Funktion der CRT-Parameter m_p , d_p , m_q und d_q , sowie aus den Faktoren des RSA-Moduls n , p und q berechnet (allgemein)?
- Wie ändert sich diese Formel wenn $(p^{-1} \bmod q) \cdot m_q^{d_q}$ durch X ersetzt wird?
- Angenommen man könnte die Berechnung der Signatur auf der Smartcard stören. Das Ergebnis sei dadurch wie zuvor beschrieben zu s' verändert. Was ergibt sich für

$$ggT(s - s', n)?$$

- Wie lässt sich dies für einen Angriff nutzen?

Schritt 2 Greifen Sie die Berechnung der Signatur auf der Smartcard wie diskutiert an. Dokumentieren Sie alle dafür durchgeführten Signaturberechnungen der Smartcard mit einem Screenshot!

In wirklichen Angriffen könnten unterschiedliche Störungen wie ein elektromagnetisches Feld, Licht- oder Hitzeeinwirkung oder Spannungsschwankungen dazu genutzt werden, einen solchen Fehler herbeizurufen [?].

3.3.3 Bug-Attack

Adi Shamir wies Ende 2007 in einer wissenschaftlichen Notiz [?] auf eine möglicherweise bedrohliche Sicherheitslücke bei der Durchführung von kryptografischen Berechnungen auf Prozessoren hin, die für bestimmte Eingabewerte falsche Ergebnisse z.B. bei der Multiplikation liefern. Seine Beobachtung dabei ist, dass die Anzahl potentieller Bugs mit steigender Wortbreite der Prozessoren zunimmt.²

Für den Fall, dass ein Prozessor z.B. bei der Multiplikation zweier Speicherwörter a und b ein falsches Ergebnis res' liefert, sollen gerade auf dem diskreten Logarithmus basierende Berechnungen geknackt werden können, deren Lösung bei der Wahl einer geeigneten Gruppe ansonsten als schwer gilt. Auch das weit verbreitete RSA-Verfahren sei somit betroffen.

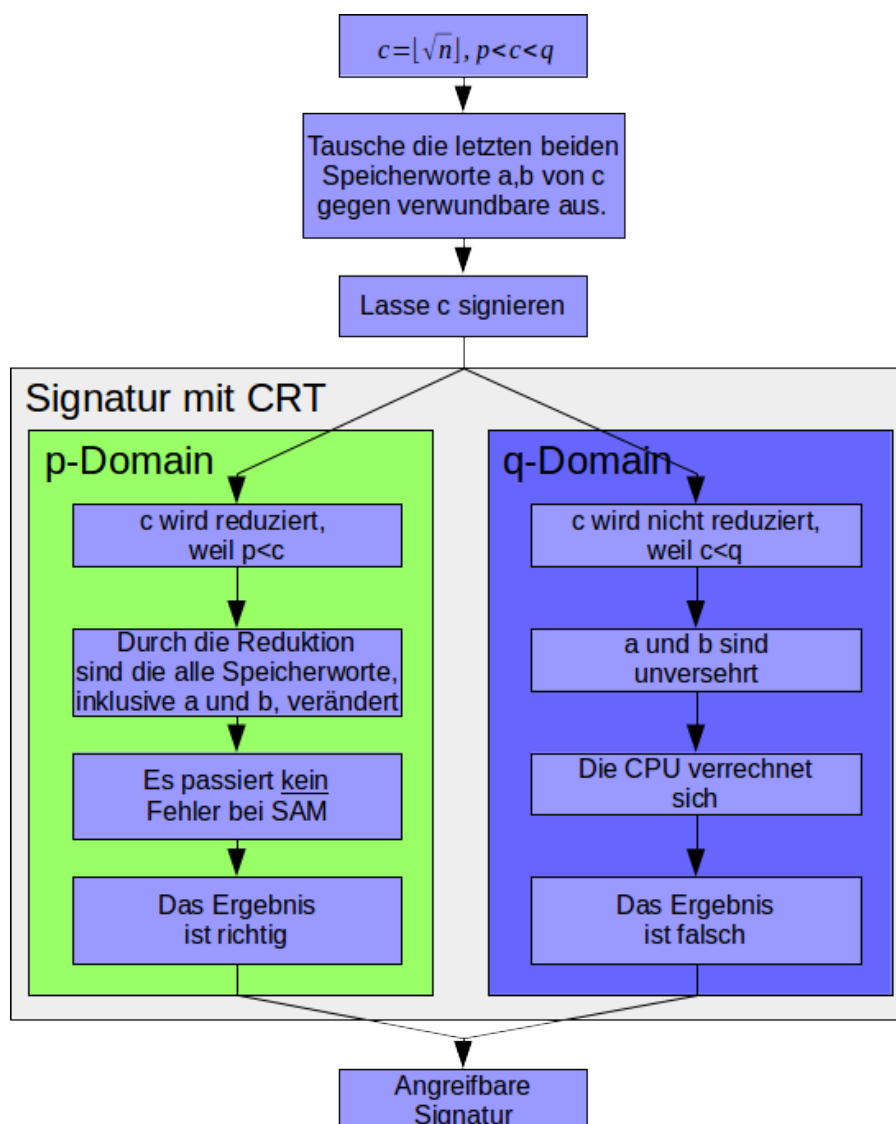


Abbildung 2: Der Ablauf einer Signatur, die mit einer Bug Attack angreifbar ist.

Auf diese Weise kann, analog zu herkömmlichen Fault-Attacks bspw. der geheime Schlüssel der zur Berechnung einer Signatur mit RSA-CRT verwendet wurde, berechnet werden. Die bahnbrechen-

²Lange Zahlen werden auf modernen Prozessoren multipliziert, indem sie in Wörter der Wortbreite des Prozessors zerlegt und die Wörter der Faktoren, analog zur schriftlichen Multiplikation, mit Hilfe der in der ALU zur Verfügung stehenden Arithmetik paarweise miteinander multipliziert werden. Das Ergebnis entsteht durch Addition der Teilergebnisse.

de Neuerung ist allerdings, dass der Angreifer keinerlei physikalische Kontrolle über das Zielsystem benötigt.

Im schlimmsten Fall schickt der Angreifer eine einzige Signaturanfrage über ein öffentliches Netz an das Zielsystem (ggf. auch ein normaler PC) und kann aus der Antwort unmittelbar den geheimen Schlüssel berechnen. Dazu muss er lediglich wissen, dass in dem Zielsystem ein anfälliger Chip verbaut ist, um einen entsprechenden Angriffsvektor wie folgt (Bsp. RSA-CRT) konstruieren zu können:

1. Der Angreifer sucht sich mit Hilfe des öffentlichen Schlüssels ein c aus mit $p < c < q$
 - Ansatz: ganzzahliges c in der Nähe von \sqrt{n}
2. Der Angreifer wählt ein m wie folgt (Darstellung in Speicherwörtern):

$c :=$
 \swarrow

\$...	\$	\$	\$
----	-----	----	----	----

$m :=$
 \swarrow

\$...	\$	a	b
----	-----	----	---	---

Anmerkung: Die mit „\$“ gekennzeichneten Speicherwörter können frei gewählt werden, müssen aber ungleich a, b sein!

3. Der Angreifer sendet m zur Signatur an das Zielsystem
4. Der Angreifer berechnet einen der Faktoren $q = \text{ggT}(s^e - m, n)$, Erläuterung: siehe unten!

Die Idee dabei ist, dass bei der Berechnung der Signatur s mit Hilfe des chinesischen Restsatzes die Nachricht m zunächst aufgeteilt wird in m_p und m_q (siehe oben!). Da c in der Nähe von \sqrt{n} gewählt wurde, und m sich nur in den beiden niederwertigsten Speicherwörtern von c unterscheidet, nimmt man an (es ist essentiell für den Erfolg des Angriffs, kann aber bei der Konstruktion nicht garantiert werden), dass m modulo des kleineren Faktors p zunächst reduziert und damit im Laufe der weiteren Berechnungen pseudozufällige Elemente aus \mathbb{Z}_p auftreten. Im Allgemeinen treten also hier keine Fehler mehr auf.

Anders sieht es aus modulo des größeren Faktors q . Hier wird das präparierte m nicht reduziert, da es kleiner ist als q . Folglich wird in der ersten Iteration des Square-and-Multiply-Algorithmus zur Exponentiation m mit sich selbst multipliziert. Da m länger ist als die Wortbreite des Prozessors, werden die Teilwörter paarweise miteinander multipliziert; unter anderem also auch die Werte a und b , was zu einem Berechnungsfehler und somit zu einem pseudozufälligen Ergebnis führt.

Das Resultat ist, dass die Signatur s zwar modulo p korrekt gebildet wurde, aber modulo q falsch ist und somit analog zum klassischen Fault-Attack einer der Faktoren von n berechnet werden kann. Dass dies mit Hilfe der Formel (4) möglich ist, kann man sich mit Hilfe einiger Rechenregeln für den ggT [?], sowie der Tatsache, dass $m = m_q + \alpha \cdot q$ und $\text{ggT}(X, p) = 1$ (mit großer Wahrscheinlichkeit) gilt, wie folgt klarmachen:

Grundlagenpraktikum NDS - Angriffe auf RSA 10

15. März 2020

$$ggT(s^e - m, n) = ggT(X \cdot q + (p^{-1} \bmod q) \cdot m_q^{d_q} \cdot p, n) \quad (1)$$

$$= ggT(X \cdot q + (p^{-1} \bmod q) \cdot m_q^{d_q} \cdot p, p) \cdot ggT(X \cdot q + (p^{-1} \bmod q) \cdot m_q^{d_q} \cdot p, q) \quad (2)$$

$$= ggT(X \cdot q, p) \cdot ggT(X \cdot q + (p^{-1} \bmod q) \cdot m_q^{d_q} \cdot p, q) \quad (3)$$

$$= 1 \cdot ggT((p^{-1} \cdot p \cdot m_q^{d_q})^e - m, q) \quad (4)$$

$$= 1 \cdot ggT(m_q - (m_q + \alpha q), q), \text{ für ein } \alpha \in \mathbb{Z} \quad (5)$$

$$= 1 \cdot ggT(-\alpha q, q) = 1 \cdot q = \mathbf{q} \quad (6)$$

Annahme: Ein Prozessor mit der Wortbreite (für unser Beispiel) von 4 Bit berechnet Signaturen mit Hilfe des RSA-CRT. Sie wissen, dass dieser Prozessor für die Multiplikation der Speicherwörter (4; 11) bzw. (11; 4) fehlerhafte, pseudozufällige Werte liefert. Sie sollen nun das simulierte Zielsystem durch Konstruktion eines geeigneten Angriffsvektors angreifen.

Führen Sie die folgenden Schritte durch und beantworten Sie die Fragen. Dokumentieren Sie den Angriff selbst durch einen Screenshot. Die EEA-Berechnungen dürfen auch hier wieder mittels Hilfsmitteln berechnet werden, aber dennoch sollte das Verständnis für die einzelnen Schritte jedoch deutlich hervorgehen.

- Besuchen sie mithilfe eines Webbrowsers die lokale URL <http://192.168.1.6/bugattack>.
- Wie lauten öffentlicher Schlüssel und Modulus des Zielsystems?
- Konstruieren Sie ein geeignetes m .
- Erläutern Sie anhand der binären Darstellung, warum dieses m geeignet ist.
- Schicken Sie Ihr m in Dezimaldarstellung an das Zielsystem. Können Sie mithilfe der Webapplikation allein erkennen, dass sich ein m für den Angriff eignet?
- Wie lautet die Faktorisierung von n ?
- Wie hoch ist die Wahrscheinlichkeit, ein fehlerhaft berechnetes Paar (a, b) auf einem 4-Bit-Prozessor zu finden? Vergleichen Sie dies mit der Wahrscheinlichkeit auf einem 64-Bit-Prozessor. Diskutieren Sie die Gefährlichkeit einer solchen Sicherheitslücke.

Eine beispielhafte Eingabe in die Webapplikation *bugattack*:

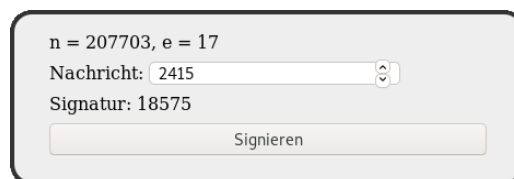


Abbildung 3: Webapplikation *bugattack*

Hinweis: Bei der Berechnung der Signatur kommt die oben diskutierte Arithmetik des Prozessors zur Anwendung, mit der Ausnahme, dass zufällige Fehler modulo p ausgeschlossen sind, um Ihnen den Angriff etwas zu erleichtern.

3.4 Kontrollfragen

1. Wie wird ein RSA-Schlüsselpaar erzeugt?
2. Wie wird mit RSA verschlüsselt? Wie wird wieder entschlüsselt?
3. Wie verschlüsselt man lange Nachrichten, die nicht in \mathbb{Z}_n liegen?
4. Wie wird eine RSA-Signatur berechnet? Wie wird diese verifiziert?
5. Wozu dient eine Hashfunktion bei der Signaturberechnung?
6. Welche öffentlichen Exponenten e werden häufig verwendet?
7. Warum gerade diese Exponenten? (Tipp: Binärdarstellung)
8. Angenommen, A möchte mit B verschlüsselt kommunizieren. Wie kann ein Angreifer die Vertraulichkeit des Systems angreifen, wenn A und B vorab noch nicht den Public Key des jeweils anderen kennen?
9. Abstrahieren sie! Der Angriff aus Kontrollfrage 8 ist möglich, weil welches Sicherheitsziel nicht von den Public Keys erfüllt wird?
10. Wie könnte man das Sicherheitsziel aus Kontrollfrage 9 erfüllen?
11. Angenommen, sie kennen zwei RSA-Signaturen s_1 und s_2 . Wie können sie eine gültige, dritte Signatur ohne Kenntnis von d berechnen?
12. Beschreiben Sie grob, wie ein Angreifer vorgehen kann, um ein geheimes m zu entschlüsseln, wenn es mit $e = 3$ an drei Empfänger verschlüsselt wurde!
13. Wozu nutzt man den chinesischen Restsatz in einigen RSA-Implementierungen?
14. Was ist eine Fault Attack? Skizzieren Sie, wie eine Fault Attack gegen RSA-CRT funktioniert!
15. Was macht man sich bei der Bug Attack zunutze?