



Lösungsskizzen zu Übung 10

Achtung: Die Lösungsskizzen sollen nur die Ideen veranschaulichen und sind daher gelegentlich sehr informell geschrieben; präzisere Lösungen sollten allerdings leicht daraus ableitbar sein.

Aufgabe 1 (RSA-OAEP-Variante)

Die Variante ist *nicht* CCA-IND. Wir betrachten zunächst den Fall ohne $0^{n/4}$. In diesem Fall kann ein Angreifer die Nachrichten $m_0 = 000 \cdots 0$ und $m_1 = 001 \cdots 1$ an die Verschlüsselungsbox schicken, und dann den erhaltenen Ciphertext C zu $C^* = C \cdot 2^e \bmod N$ verändern und an die Entschlüsselungsbox schicken. Da die Multiplikation mit 2^e die Bits von $m_b || r$ um eine Position nach links verschiebt (einen Übertrag gibt es wegen der führenden Nullen nicht), erhält der Angreifer so die Antwort $00 \cdots 0 || \text{msb}(r)$ oder $01 \cdots 1 || \text{msb}(r)$. Daraus kann er leicht das Bit b bestimmen.

Im Fall mit den “Test-Bits” $0^{n/4}$ funktioniert ein entsprechender Angriff fast analog. Es kann dabei lediglich passieren, dass das erste Bit $\text{msb}(r)$ von r den Wert 1 hat, und somit ein 1-Bit in die Test-Bits geschoben wird. In diesem Fall erhält der Angreifer nur die Antwort \perp von der Entschlüsselungsbox. Ist das erste Bit dagegen 0, kann der Angreifer aus der Antwort das Bit b einfach bestimmen (mit Wahrscheinlichkeit 1). Im Fall \perp lassen wir den Angreifer dann einfach mit Wahrscheinlichkeit $\frac{1}{2}$ das Bit b erraten, indem er ein zufälliges Bit a ausgibt. Da die Fälle $\text{msb}(r) = 0/1$ jeweils mit Wahrscheinlichkeit $1/2$ auftreten, gilt

$$\text{Prob}[a = b] = \text{Prob}[\text{msb}(r) = 0] \cdot 1 + \text{Prob}[\text{msb}(r) = 1] \cdot \frac{1}{2} = \frac{3}{4}.$$

Bemerkung: Die “Bit-Schiebereien” funktionieren beim original RSA-OAEP nicht, da die Werte noch durch die (quasi zufälligen) Funktionen G und H verarbeitet werden. Solche “kleinen” Änderungen wie oben erzeugen daher eine “unkontrollierte” Änderung aller Bits, insbesondere in den Test-Bits.

B.W.

Aufgabe 2 (Kombination von Hash-Funktionen)

Die erste Variante H erreicht tatsächlich das gewünschte Sicherheitsniveau. Jede Kollision $m \neq m^*$ für H bedeutet wegen $H(m) = H_0(m) || H_1(m) = H_0(m^*) || H_1(m^*) = H(m^*)$ und der Längeninvarianz, dass auch $H_0(m) = H_0(m^*)$ und $H_1(m) = H_1(m^*)$ gelten muss. Daher ist eine solche Kollision automatisch auch eine Kollision für beide Hash-Funktionen H_0, H_1 , und –sofern eine der beiden Funktionen sicher ist— kann man solche Kollisionen folglich nicht effizient finden.

Für den interessierten Leser: Ohne die Anforderung an die Längeninvarianz gilt die Aussage im Allgemeinen nicht mehr. In diesem Fall kann man sogar zwei kollisionsresistente Hash-Funktionen H_0, H_1 angeben, so dass H trotzdem unsicher wird. Beispiel: Sei H' eine beliebige, kollisionsresistente Hash-Funktion. Definiere H_0 durch $H_0(0) = 0$, $H_0(1) = 01$ und $H_0(m) = 1 || H'(m)$ sonst ($m \notin \{0, 1\}$), sowie H_1 durch $H_1(0) = 11$, $H_1(1) = 1$ und $H_1(m) = 0 || H'(m)$ sonst ($m \notin \{0, 1\}$). Dann sind beide Funktionen jeweils kollisionsresistent (da H' sicher ist), aber $H(0) = H_0(0) || H_1(0) = 011 = H_0(1) || H_1(1) = H(1)$, also ist das Paar $(0, 1)$ eine leicht zu findende Kollision für H .

Die zweite Variante H^* dagegen erfüllt die Anforderung nicht. Angenommen, H_1 bleibt sicher, aber man kann leicht Kollisionen $m \neq m^*$ in H_0 finden. Dann ist aber eine solche Kollision auch Kollision für H^* , da $H^*(m) = H_1(H_0(m)) = H_1(H_0(m^*)) = H^*(m^*)$.

Ob man H^* auch erfolgreich angreifen kann, wenn H_1 unsicher wird, aber H_0 sicher bleibt, hängt wesentlich davon ab, wie die auffindbaren Kollisionen für H_1 aussehen und ob man sie “durch H_0 hindurch” zurückrechnen kann auf Werte m, m^* .