

BDD-basierte Angriffe auf Flusschiffren

Dirk Stegemann

`dirk.stegemann@uni-mannheim.de`

Lehrstuhl für Theoretische Informatik
Universität Mannheim

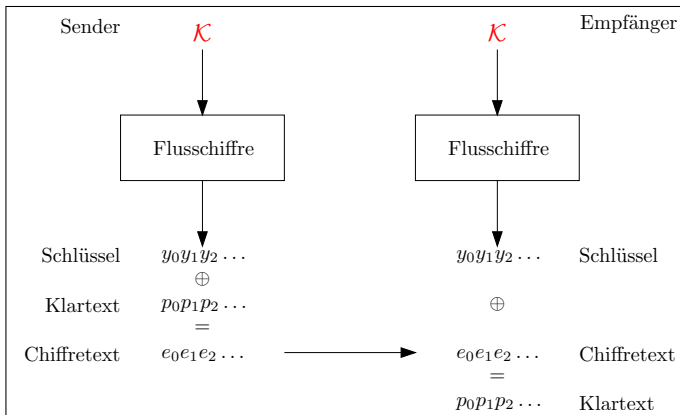
Metarheinmain Chaos Days 11b

2.- 4. September 2005

Überblick

- 1 Einführung
 - Flusschiffren
 - Bluetooth-Chiffre E_0
 - GSM-Chiffre A5/1
- 2 Idee des Angriffs
- 3 Binary Decision Diagrams
 - Definitionen
 - Operationen auf FBDDs
 - Implementationen und Anwendungen
- 4 Kryptanalyse mit BDDs
 - BDD-Algorithmus
 - Ergebnisse
- 5 Zusammenfassung

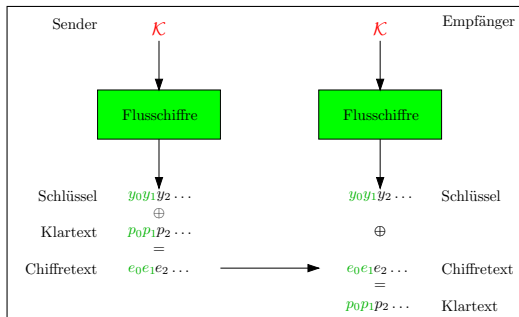
Funktionsweise



Angriffsmodell

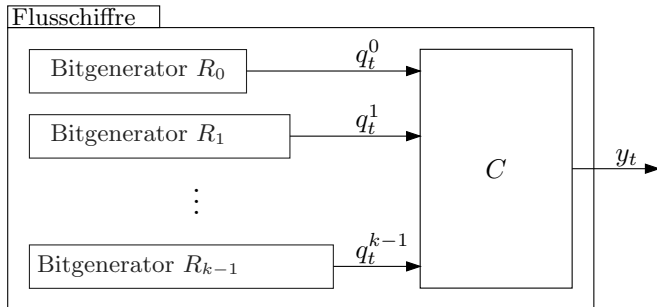
Angreifer kennt

- **Aufbau** der Flusschiffre
- Klartext-Chiffretext Paare (p_i, c_i) , $i \in \mathcal{I}$
→ Schlüsselbits y_i



Ziel des Angreifers: Finde den **geheimen Schlüssel \mathcal{K}**

Interner Aufbau

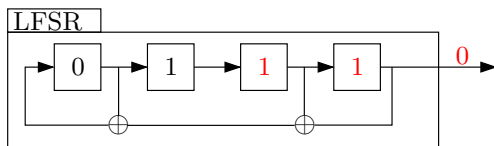


- Bitgeneratoren R_0, \dots, R_{k-1}
- (nichtlineare) Kompressionsfunktion C

LFSR als Bitgeneratoren

LFSR = **L**inear **F**eedback **S**hift **R**egister

- n Registerzellen
- Rückkopplungskanal
- **Initialzustand**
(q_0, \dots, q_{n-1})
- 1 Bit Ausgabe in jedem Takt

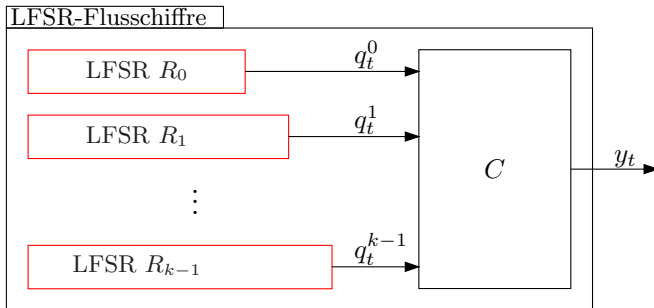


Beispiel: (q_1, \dots, q_4) = (1, 0, 1, 1)

Beobachtung

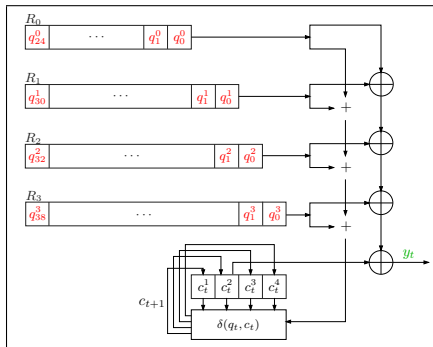
- *Alle Bits q_t sind Linearkombinationen des Initialzustands.*

LFSR-basierte Flusschiffren



geheimer Schlüssel \mathcal{K} = Initialzustände der LFSR

Die E_0 Flussschiffre



In jeder Iteration t :

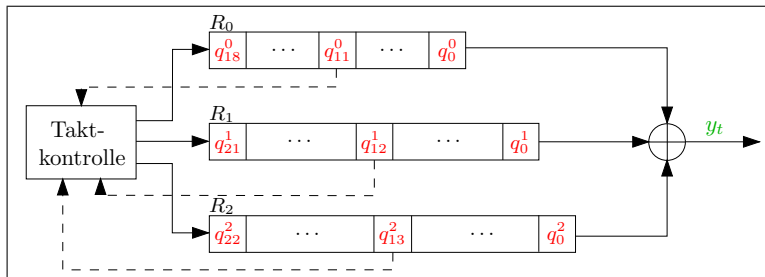
- Erzeuge Schlüsselbit

$$y_t = q_t^0 \oplus \dots \oplus q_t^3 \oplus c_t^2$$
- Aktualisiere Zustandsbits

$$c_{t+1} = \delta(q_t, c_t)$$
- Takte alle 4 LFSR

Die A5/1 Flusschiffre

LFSR R_0 , R_1 , R_2 mit Kontrollpositionen N_0 , N_1 , N_2

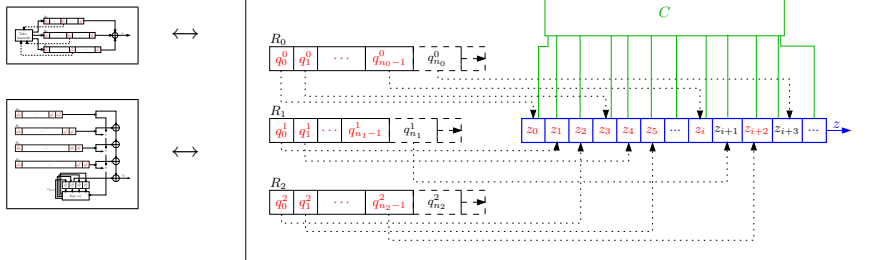


In jeder Iteration t :

- Erzeuge Schlüsselbit $y_t = q_t^0 \oplus q_t^1 \oplus q_t^2$
- Takte R_j gdw. $q_{N_j+t} = \text{maj}_3(q_{N_0+t}^0, q_{N_1+t}^1, q_{N_2+t}^2)$

- 1 Einführung
 - Flusschiffren
 - Bluetooth-Chiffre E_0
 - GSM-Chiffre A5/1
- 2 **Idee des Angriffs**
- 3 Binary Decision Diagrams
 - Definitionen
 - Operationen auf FBDDs
 - Implementationen und Anwendungen
- 4 Kryptanalyse mit BDDs
 - BDD-Algorithmus
 - Ergebnisse
- 5 Zusammenfassung

Generische Darstellung für E_0 und A5/1



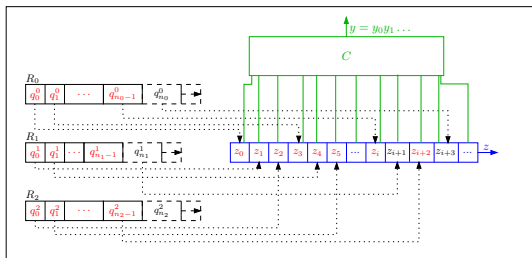
- LFSR erzeugen **internen Bitstrom z**
- **Kompressionsfunktion C** berechnet Schlüsselstrom $y = C(z)$

Kryptanalyseansatz

Problem: Gegeben ein Präfix des **Schlüsselstroms** y , berechne den **Initialzustand** x der LFSR.

Beobachtung

Der **Initialzustand** x ist in den ersten Bits des **internen Bitstroms** z enthalten.

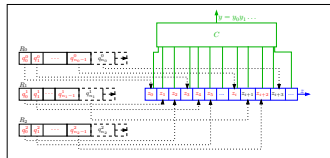


→ Strategie: Rekonstruiere aus y den **internen Bitstrom** z .

Rekonstruktion des internen Bitstroms z

Konsistenzbedingungen für z

- 1 z erfüllt die linearen Relationen der LFSR.
- 2 $C(z)$ ist Präfix des beobachteten Schlüsselstroms y .



Algorithmus

Berechne für $m = 1, 2, \dots$ die Menge

$$\begin{aligned}
 \mathcal{P}_m &= \{z \in \{0, 1\}^m \mid z \text{ erfüllt (1)}\} \cap \{z \in \{0, 1\}^m \mid z \text{ erfüllt (2)}\} \\
 &= \mathcal{P}_{m-1} \times \{0, 1\} \\
 &\quad \cap \{(z_0, \dots, z_{m-1}) \in \{0, 1\}^m \mid z_{m-1} \text{ erfüllt (1)}\} \\
 &\quad \cap \{z \in \{0, 1\}^m \mid z \text{ erfüllt (2)}\}
 \end{aligned}$$

bis nur noch mit y vollständig konsistente Bitströme übrig sind.

Rekonstruktion des internen Bitstroms z

Ansatz benötigt effiziente Datenstruktur zur Repräsentation von Teilmengen aus $\{0, 1\}^m$.

- Repräsentiere $F \subseteq \{0, 1\}^m$ durch die charakteristische Boolesche Funktion

$$f : \{0, 1\}^m \rightarrow \{0, 1\}$$
$$z \mapsto \begin{cases} 1 & z \in F \\ 0 & z \notin F \end{cases}$$

- Suche eine effiziente Datenstruktur für f .

→ *Binary Decision Diagrams (BDDs)*

- 1 Einführung
 - Flusschiffren
 - Bluetooth-Chiffre E_0
 - GSM-Chiffre A5/1
- 2 Idee des Angriffs
- 3 Binary Decision Diagrams
 - Definitionen
 - Operationen auf FBDDs
 - Implementationen und Anwendungen
- 4 Kryptanalyse mit BDDs
 - BDD-Algorithmus
 - Ergebnisse
- 5 Zusammenfassung

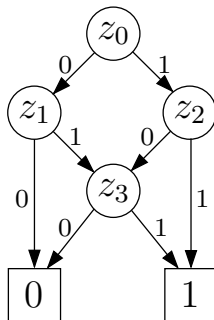
Binary Decision Diagram (BDD)

gerichteter, azyklischer Graph G_f

- 1 Quelle und 2 Senken
 - innere Knoten haben $outdeg = 2$
 - Knotenbeschriftungen
 - Senken: 0 bzw. 1
 - innere Knoten: Variablen aus $Z_n = \{z_0, \dots, z_{n-1}\}$
 - Kantenbeschriftungen $\in \{0, 1\}$
-
- Jedes $z \in \{0, 1\}^n$ definiert Pfad in G_f
 - G_f repräsentiert Funktion f mit $f(z) :=$ Label der erreichten Senke,

z.B.

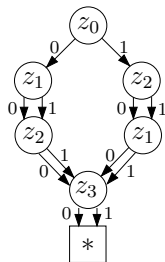
(z_0, z_1, z_2, z_3)	$f(z)$
$(0, 1, 1, 0)$	0
$(1, 0, 0, 1)$	1



Steuerungsgraph

BDD G_0 über Z_n

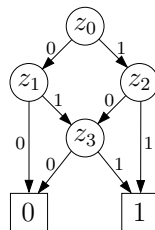
- nur eine Senke
- auf jedem Pfad alle Variablen $z_i \in Z_n$ genau einmal eingelesen



Free Binary Decision Diagram (G_0 -FBDD)

BDD G_f über Z_n

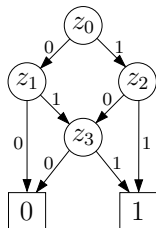
- definiert bzgl. Steuerungsgraph G_0
- Für jedes $z \in \{0, 1\}^n$ wird durch G_0 gegebene Einlesereihenfolge respektiert.



Wichtige Struktureigenschaft von FBDDs:

Read-once Eigenschaft

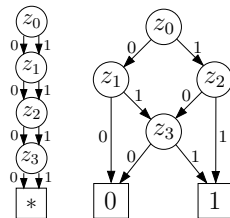
In einem G_0 -FBDD wird auf allen Pfaden jede Variable z_i höchstens einmal eingelesen.



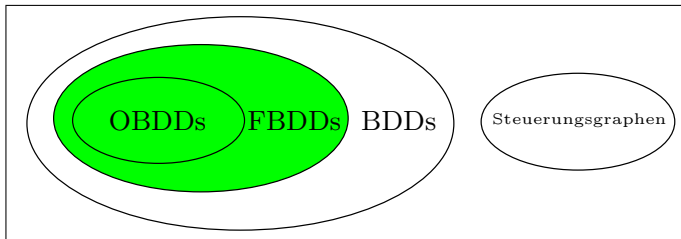
Ordered Binary Decision Diagram (OBDD)

FBDD G_f mit Steuerungsgraph G_0

- G_0 ist zur Liste entartet



Insgesamt:

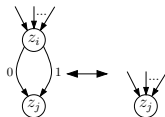


ausgewählte Operationen auf FBDDs

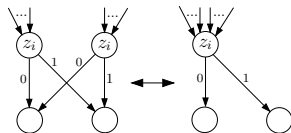
- Minimierung
- Evaluation
- SAT-ENUM
- Synthese
- Äquivalenztest

Minimierung

Eingabe	G_0 -FBDD G_f
Problem	Bestimme minimalen G_0 -FBDD G_f^* für f



elimination rule



merging rule

Satz

G_f ist genau dann minimal, wenn weder die merging rule noch die elimination rule anwendbar ist.

Wende die Regeln bottom-up an
 Laufzeit und Speicher $O(|G_f|)$

Minimierung

Eigenschaften eines minimierten G_0 -FBDD G_f^*

- 1 Die Größe ist beschränkt durch

$$|G_f^*| \leq n \cdot |One(f)|$$

mit $One(f) = \{a \in \{0, 1\}^n \mid f(a) = 1\}$

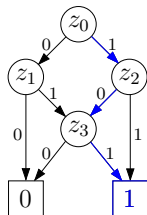
- 2 Für fest gewähltes G_0 ist G_f^* eindeutig bestimmt.

Evaluation

Eingabe G_0 -FBDD G_f

$a \in \{0, 1\}^n$

Problem Berechne $f(a)$



*Verfolge den durch a
definierten Pfad bis zu
einer Senke*

*Laufzeit und Speicher
 $O(n)$*

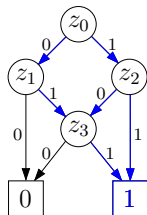
SAT-ENUM

Eingabe G_0 -FBDD G_f

Problem Finde alle $a \in \{0,1\}^n$
mit $f(a) = 1$

- 1 *Minimiere G_f*
- 2 *Tiefensuche nach der 1-Senke*
- 3 *Belege freie Variablen*

Laufzeit und Speicher
 $O(n \cdot |\text{One}(f)|)$

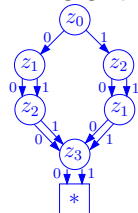
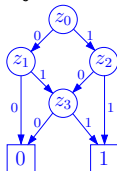
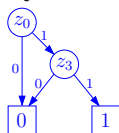


Synthese

Eingabe G_0 -FBDDs G_f, G_g
 $\otimes : \{0, 1\}^2 \rightarrow \{0, 1\}$

Problem Berechne $G_{f \otimes g}$

- 1 *Simultane Tiefensuche in G_0, G_f und G_g*
- 2 *Wenn Senken erreicht, berechne Funktionswert als $(f \otimes g)(a) = f(a) \otimes g(a)$*

Steuerungsgraph G_0  G_0 -FBDD G_f  G_0 -FBDD G_g 

Synthese

Implementation:

- *computed table* speichert Berechnungsergebnisse
- *unique table* speichert einen Repräsentantenknoten für jedes Tripel (z_i , 0-Nachfolger, 1-Nachfolger)

→ Synthese mit integrierter Minimierung

Laufzeit und Speicherplatz $O(|G_0| \cdot |G_f| \cdot |G_g|)$

Äquivalenztest

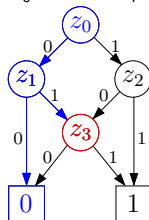
Eingabe G_0 -FBDDs G_f, G_g

Problem Entscheide, ob $f \equiv g$

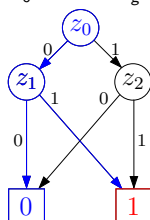
- 1 *Minimiere G_f und G_g*
- 2 *Äquivalenztest durch simultane Tiefensuche in G_f und G_g*

Laufzeit und Speicher
 $O(|G_f| + |G_g|)$

G_0 -FBDD G_f



G_0 -FBDD G_g



Implementationen

Viele OBDD-Pakete verfügbar

- kommerzielle Pakete
 - EHV (IBM Research)
 - CMU (Carnegie Mellon University)
- frei verfügbare Pakete
 - CAL (University of California)
 - BuDDy (IT-University of Copenhagen)
 - CUDD (University of Colorado)

FBDD-Paket auf Basis von CUDD (Universität Mannheim)

Anwendungen

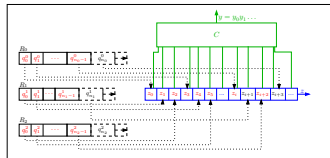
- Formale Schaltkreisverifikation
 - ① Erzeuge FBDDs
 - G_{opt} für optimierten Schaltkreis
 - G_{ref} für Referenzschaltkreis
 - ② Führe Äquivalenztest für G_{opt} und G_{ref} durch
- Kryptanalyse von Flusschiffren

- 1 Einführung
 - Flusschiffren
 - Bluetooth-Chiffre E_0
 - GSM-Chiffre A5/1
- 2 Idee des Angriffs
- 3 Binary Decision Diagrams
 - Definitionen
 - Operationen auf FBDDs
 - Implementationen und Anwendungen
- 4 **Kryptanalyse mit BDDs**
 - BDD-Algorithmus
 - Ergebnisse
- 5 Zusammenfassung

Rekonstruktion des internen Bitstroms z

Konsistenzbedingungen für z

- ① z erfüllt die linearen Relationen der LFSR.
- ② $C(z)$ ist Präfix des beobachteten Schlüsselstroms y .



FBDD-Algorithmus

Berechne für $m = 1, 2, \dots, m^*$ die G_0 -FBDDs

$$P_m = P_{m-1} \wedge \underbrace{\{z \in \{0, 1\}^m \mid z_{m-1} \text{ erfüllt (1)}\}}_{S_m} \\ \wedge \underbrace{\{z \in \{0, 1\}^m \mid z \text{ erfüllt (2)}\}}_{Q_m}$$

bis nur noch mit y vollständig konsistente Bitströme übrig sind.

Rekonstruktion von z mit FBDDs

Laufzeit des Algorithmus hängt ab von

- m^* = Anzahl der Iterationen
- Laufzeit der Syntheseoperationen
→ maximale Größe der Zwischenergebnisse $|P_m|$
- Parameter der Flusschiffre
 - Informationsrate $\alpha \in (0, 1]$
 - (best case) Kompressionsrate $\gamma \in (0, 1]$

Satz (Krause, EUROCRYPT 2002)

Für Flusschiffren der Schlüssellänge n mit $|G_0| \in n^{O(1)}$ und $|Q_m| \in m^{O(1)}$ gilt

- $m^* = \lceil \alpha^{-1} n \rceil$
- $|P_m| \leq n^{O(1)} 2^{\frac{1-\alpha}{1+\alpha} n}$ für $m \leq m^*$

Rekonstruktion von z mit FBDDs

Folgerung

Der interne Bitstrom z (und damit der *geheime Initialzustand* x) kann in einer Laufzeit und mit einem Speicherplatzbedarf von

$$n^{O(1)} 2^{\frac{1-\alpha}{1+\alpha} n}$$

aus dem *beobachteten Schlüsselstrom* rekonstruiert werden.

Wie viele Schlüsselbits brauchen wir für die Rekonstruktion von x ?

Beobachtung

Für die Rekonstruktion von x genügen die ersten $\lceil \gamma \alpha^{-1} n \rceil$ aufeinanderfolgenden Bits von y .

Anwendung auf A5/1 und E_0

Was bedeutet dieses Resultat für A5/1 und E_0 ?

	n	α	γ
A5/1	64	0,2193	0,25
E_0	128	0,25	0,25

Folgerung

Der **geheime Initialzustand x** kann in Laufzeit und mit Speicherplatz

- $n^{O(1)} 2^{0,6403n} c_1 \cdot 2^{41}$ aus den ersten $\lceil 1.14n \rceil$ 73 Bits von y für A5/1
- $n^{O(1)} 2^{0,6n} c_0 \cdot 2^{77}$ aus den ersten n 128 Bits von y für E_0

berechnet werden.

Beachte: Brute-force Testen aller x dauert $n^{O(1)} 2^n c_i \cdot 2^{64}$ bzw. 2^{128} .

BDD-Angriff in der Praxis...

... mit realen Schlüssellängen

128 Bit Schlüsselstrom, d.h. 128 Klartext-Chiffretext Paare

→ realistisch

A5/1 Generator

- $c_1 \cdot 2^{41}$ Operationen eventuell praktikabel
- $c_1 \cdot 2^{41} \cdot 16$ Byte = $c_1 \cdot 32$ Terabyte Speicher möglich?

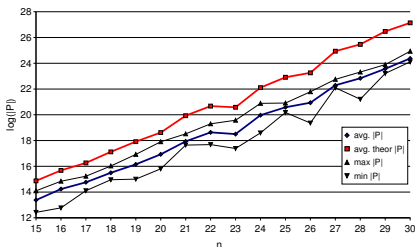
E_0 Generator

- $c_0 \cdot 2^{77}$ Operationen (noch) nicht praktikabel
- $c_0 \cdot 2^{77} \cdot 16$ Byte = $c_0 \cdot 2^{42}$ Terabyte Speicher unrealistisch

BDD-Angriff in der Praxis...

...auf einem Pentium IV System mit 4 GB Hauptspeicher

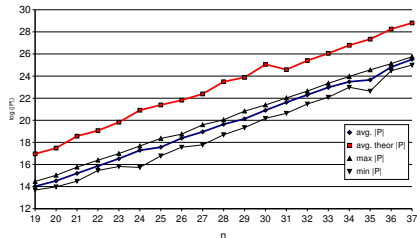
A5/1 Generator



für $n = 30$ durchschnittlich

- 65 Minuten
- 338 MB Speicher für P_m

E0 Generator



für $n = 37$ durchschnittlich

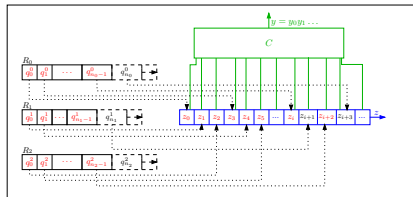
- 87 Minuten
- 743 MB Speicher für P_m

- 1 Einführung
 - Flusschiffren
 - Bluetooth-Chiffre E_0
 - GSM-Chiffre A5/1
- 2 Idee des Angriffs
- 3 Binary Decision Diagrams
 - Definitionen
 - Operationen auf FBDDs
 - Implementationen und Anwendungen
- 4 Kryptanalyse mit BDDs
 - BDD-Algorithmus
 - Ergebnisse
- 5 Zusammenfassung

Zusammenfassung

Voraussetzungen

- ① Darstellung in der Form $y = C(z)$
- ② Größen von Steuerungsgraph und Schlüsselstrom-FBDD polynomiell in n



Ergebnisse

- anwendbar u.a. für A5/1 Generator und E_0 Generator
- Speicherplatz ist Engpassfaktor
- (momentan noch) keine Gefahr für praktisch verwendete Schlüssellängen

Vielen Dank!

`dirk.stegemann@uni-mannheim.de`