Cryptanalysis of Symmetric-Key Primitives: Automated Techniques

Nicky Mouha

ESAT/COSIC, KU Leuven, Belgium
IBBT, Belgium

Summer School on Tools, Mykonos
Tuesday, May 29, 2012
Symmetric-key Ciphers: Types of attacks

- Statistical attacks
  - Linear and differential cryptanalysis, slide attacks,...
  - Detect statistical non-randomness
Symmetric-key Ciphers: Types of attacks

- **Statistical attacks**
  - Linear and differential cryptanalysis, slide attacks,...
  - Detect statistical non-randomness

- **Meet-in-the-middle attacks**
  - Many techniques (splice-and cut, partial matching, partial fixing,...), guess-and-determine attacks, attack on 2DES,...
  - Separate equations into two or more groups to solve them more efficiently
Symmetric-key Ciphers: Types of attacks

- **Statistical attacks**
  - Linear and differential cryptanalysis, slide attacks,...
  - Detect statistical non-randomness
- **Meet-in-the-middle attacks**
  - Many techniques (splice-and-cut, partial matching, partial fixing,...), guess-and-determine attacks, attack on 2DES,...
  - Separate equations into two or more groups to solve them more efficiently
- **Algebraic attacks**
  - See next slide
Algebraic Attacks: Definition

- Represent cryptographic primitive as system of equations
- Use equation solver to retrieve key
Algebraic Attacks: Definition

- Represent cryptographic primitive as system of equations
- Use equation solver to retrieve key
  - SAT solvers
    - MiniSat2, CryptoMiniSat,...
  - Gröbner basis method
    - Buchberger’s algorithm, $F_4$, $F_5$,...
  - Mixed Integer Linear Programming (MILP)
    - CPLEX, SYMPHONY,...
Algebraic Attacks: Definition

- Represent cryptographic primitive as system of equations
- Use equation solver to retrieve key
  - SAT solvers
    - MiniSat2, CryptoMiniSat, ...
  - Gröbner basis method
    - Buchberger’s algorithm, $F_4$, $F_5$, ...
  - Mixed Integer Linear Programming (MILP)
    - CPLEX, SYMPHONY, ...
- Hopefully detects inherent structure, and
- Solves equations faster than brute force!
Algebraic Attacks: Advantages and Disadvantages

- Algebraic attacks on symmetric-key ciphers
- Biggest **disadvantages:**
  - Can only find practical attacks, no high-complexity attacks
Algebraic Attacks: Advantages and Disadvantages

- Algebraic attacks on symmetric-key ciphers
- Biggest **disadvantages**:
  - Can only find practical attacks, no high-complexity attacks
  - Execution time (and memory requirements): unpredictable
Algebraic Attacks: Advantages and Disadvantages

- Algebraic attacks on symmetric-key ciphers
- Biggest **disadvantages:**
  - Can only find practical attacks, no high-complexity attacks
  - Execution time (and memory requirements): unpredictable
  - “Not a single proper block cipher has been broken using pure algebraic techniques faster than with other techniques.” (Albrecht)
Algebraic Attacks: Advantages and Disadvantages

- **Algebraic attacks on symmetric-key ciphers**

- **Biggest **disadvantages**:**
  - Can only find practical attacks, no high-complexity attacks
  - Execution time (and memory requirements): unpredictable
  - “Not a single proper block cipher has been broken using pure algebraic techniques faster than with other techniques.” (Albrecht)

- **Biggest **advantages**:**
  - “Black box” technique, no crypto knowledge required
Algebraic Attacks: Advantages and Disadvantages

- Algebraic attacks on symmetric-key ciphers
- Biggest disadvantages:
  - Can only find practical attacks, no high-complexity attacks
  - Execution time (and memory requirements): unpredictable
  - “Not a single proper block cipher has been broken using pure algebraic techniques faster than with other techniques.” (Albrecht)

- Biggest advantages:
  - “Black box” technique, no crypto knowledge required
  - Can work with very few plaintext-ciphertext pairs
Algebraic Attacks: Advantages and Disadvantages

- Algebraic attacks on symmetric-key ciphers
- Biggest **disadvantages:**
  - Can only find practical attacks, no high-complexity attacks
  - Execution time (and memory requirements): unpredictable
  - “Not a single proper block cipher has been broken using pure algebraic techniques faster than with other techniques.” (Albrecht)

- Biggest **advantages:**
  - “Black box” technique, no crypto knowledge required
  - Can work with very few plaintext-ciphertext pairs
  - Useful to break extremely weak ciphers: Crypto-1 in 40s, HiTag2 in 6.5h on one Xeon E5345 @ 2.33GHz (Soos)
Automated Techniques: Still Useful

- Tool to construct statistical and MitM attacks
- Therefore, program execution time: not so important
Automated Techniques: Still Useful

- Tool to construct statistical and MitM attacks
- Therefore, program execution time: not so important
  - Program: executed only once
Automated Techniques: Still Useful

- Tool to construct statistical and MitM attacks
- Therefore, program execution time: not so important
  - Program: executed only once
  - More time spent on: coding, debugging, optimizing, parallel implementation, verifying,...
  - Verifying correctness: very difficult
Automated Techniques: Still Useful

- Tool to construct statistical and MitM attacks
- Therefore, program execution time: not so important
  - Program: executed only once
  - More time spent on: coding, debugging, optimizing, parallel implementation, verifying,...
  - Verifying correctness: very difficult
  - Programmer’s time: costs more than CPU time!
Automated Techniques: Still Useful

- Tool to construct statistical and MitM attacks
- Therefore, program execution time: not so important
  - Program: executed only once
  - More time spent on: coding, debugging, optimizing, parallel implementation, verifying,...
  - Verifying correctness: very difficult
  - Programmer’s time: costs more than CPU time!

- More important:
  - Easy to program
  - Easy to verify
  - Easy to parallelize
Goal of this lecture

- Use three easy, automated techniques
  - MILP programming
  - SAT solvers
  - Regular expressions

- as tools to construct attacks
  - ... and start breaking ciphers today!
Outline

1. Introduction

2. Three Easy, Automated Techniques
   - MILP Programming
   - SAT Solvers
   - Regular Expressions

3. Tools for Cryptography

4. Conclusion
Differential Cryptanalysis

Differential characteristic

\[
\begin{align*}
\Delta b &\quad \Delta c &\quad \Delta d
\end{align*}
\]
Differential Cryptanalysis: S-box

\[ S(a) \xrightarrow{\Delta \alpha} S(a \oplus \Delta \alpha) = \Delta \beta? \]

- Differential Probability \( \text{DP}(\Delta \alpha \rightarrow \Delta \beta) : \)
  \[
  \frac{\#\{0 \leq a < 2^8 : S(a) \oplus S(a \oplus \Delta \alpha) = \Delta \beta\}}{2^8}
  \]
- Max. diff. prob. (MDP): \( 4/256 = 2^{-6} \)
- AES: only component that is non-linear in \( \text{GF}(2^8) \)
- Non-active S-box: \( \text{DP}(0 \rightarrow 0) = 1 \)
- Count active S-boxes!
Representation of variables

- Every pair of bytes is “shrunk” to one bit $x_i$:
  - $x_i = 0$ if the bytes are the same
  - $x_i = 1$ if the bytes are different
Representation of variables

- Every pair of bytes is “shrunk” to one bit $x_i$:
  - $x_i = 0$ if the bytes are the same
  - $x_i = 1$ if the bytes are different
- Note: simplifies the analysis!
  - Our results prove lower bounds,
  - but characteristics may contain a contradiction
Representation of variables

- Every pair of bytes is “shrunk” to one bit $x_i$:
  - $x_i = 0$ if the bytes are the same
  - $x_i = 1$ if the bytes are different
- Note: simplifies the analysis!
  - Our results prove lower bounds,
  - but characteristics may contain a contradiction
- Next slides: focus on AES
  - but technique can analyze any cipher based on XORs,
    three-forked branches, MDS operations,...
  - Details: see Mouha et al., Inscrypt 2011
One Round of AES

\[
\begin{bmatrix}
  x_0 & x_4 & x_8 & x_{12} \\
  x_1 & x_5 & x_9 & x_{13} \\
  x_2 & x_6 & x_{10} & x_{14} \\
  x_3 & x_7 & x_{11} & x_{15}
\end{bmatrix}
\rightarrow
\begin{bmatrix}
  x_0 & x_4 & x_8 & x_{12} \\
  x_1 & x_5 & x_9 & x_{13} \\
  x_2 & x_6 & x_{10} & x_{14} \\
  x_3 & x_7 & x_{11} & x_{15}
\end{bmatrix}
\rightarrow
\begin{bmatrix}
  x_{16} & x_{20} & x_{24} & x_{28} \\
  x_{17} & x_{21} & x_{25} & x_{29} \\
  x_{18} & x_{22} & x_{26} & x_{30} \\
  x_{19} & x_{23} & x_{27} & x_{31}
\end{bmatrix}
\]

AR+SB

MC

SR
MixColumns

MDS Property:

\[ x_0 + x_5 + x_{10} + x_{15} + \]
\[ x_{16} + x_{17} + x_{18} + x_{19} \geq 5 \]

or

\[ x_0 = x_5 = x_{10} = x_{15} = \]
\[ x_{16} = x_{17} = x_{18} = x_{19} = 0 \]
MixColumns

MDS Property:
\[
\begin{align*}
x_0 + x_5 + x_{10} + x_{15} + \\
x_{16} + x_{17} + x_{18} + x_{19} & \geq 5d
\end{align*}
\]

and
\[
d = \max(x_0, x_5, x_{10}, x_{15}, x_{16}, x_{17}, x_{18}, x_{19})
\]
MixColumns

MDS Property:

\[ x_0 + x_5 + x_{10} + x_{15} + \\
 x_{16} + x_{17} + x_{18} + x_{19} \geq 5d \]

and

\[ d \geq x_0, \ d \geq x_5, \ d \geq x_{10}, \ d \geq x_{15}, \\
 d \geq x_{16}, \ d \geq x_{17}, \ d \geq x_{18}, \ d \geq x_{19} \]
Mixed-Integer Linear Programming

Mimimize
   Sum of S-box variables

Subject To
   9 equations for every MixColumns step (+1 dummy variable)
   (SubBytes, ShiftRows, add key: no equations/variables)
   Sum of plaintext variables \(\geq 1\)

Binary
   All variables / All input variables

End
Single-key AES: Bounds

<table>
<thead>
<tr>
<th># Rounds</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min. # active S-boxes</td>
<td>1</td>
<td>5</td>
<td>9</td>
<td>25</td>
<td>26</td>
<td>30</td>
<td>34</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th># Rounds</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min. # active S-boxes</td>
<td>50</td>
<td>51</td>
<td>55</td>
<td>59</td>
<td>75</td>
<td>76</td>
<td>80</td>
</tr>
</tbody>
</table>

- Using the IBM ILOG CPLEX optimizer
  - Free for academic use
- Execution time
  - no problem takes longer than 0.40 s (Intel Xeon X5670 @ 2.93GHz)
Related-key AES: Strategy

- Also one $x_i$-variable per key byte, ($x_i = 1$ iff. bytes different)
Related-key AES: Strategy

- Also one $x_i$-variable per key byte, ($x_i = 1$ iff. bytes different)
- Equations for every XOR operation:

\[
\begin{align*}
    x_{in_1} + x_{in_2} + x_{out} & \geq 2d \\
    d & \geq x_{in_1} \\
    d & \geq x_{in_2} \\
    d & \geq x_{out}
\end{align*}
\]
Related-key AES: Bounds

Minimum number of active S-boxes:

<table>
<thead>
<tr>
<th># Rounds</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES-128</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>9</td>
<td>11</td>
<td>13</td>
<td>15</td>
</tr>
<tr>
<td>AES-192</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>11</td>
</tr>
<tr>
<td>AES-256</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th># Rounds</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES-128</td>
<td>21</td>
<td>23</td>
<td>25</td>
<td>27</td>
<td>33</td>
<td>35</td>
<td>37</td>
</tr>
<tr>
<td>AES-192</td>
<td>13</td>
<td>16</td>
<td>19</td>
<td>19</td>
<td>20</td>
<td>24</td>
<td>25</td>
</tr>
<tr>
<td>AES-256</td>
<td>10</td>
<td>14</td>
<td>16</td>
<td>18</td>
<td>20</td>
<td>22</td>
<td>24</td>
</tr>
</tbody>
</table>
Related-key AES: Execution Time

- 24-core Intel Xeon X5670 @ 2.93GHz
- System used concurrently by at least 5 other people
  - ... execution times are upper bounds
Related-key AES: Execution Time

- 24-core Intel Xeon X5670 @ 2.93GHz
- System used concurrently by at least 5 other people
  - ... execution times are upper bounds
- Bounds for full AES: all less than one minute
  - except 14-round AES-256: bound in 69.84 s
Comparison to other work

- Biryukov, Nikolić (EUROCRYPT 2010, SAC 2010)
  - determine byte differences, not just zero/non-zero difference
Comparison to other work

- Biryukov, Nikolić (EUROCRYPT 2010, SAC 2010)
- determine byte differences, not just zero/non-zero difference
- AES-192, 11 rounds:
  - Biryukov, Nikolić: 31 active S-boxes, computation takes “weeks”
Comparison to other work

- Biryukov, Nikolić (EUROCRYPT 2010, SAC 2010)
- determine byte differences, not just zero/non-zero difference
- AES-192, 11 rounds:
  - Biryukov, Nikolić: 31 active S-boxes, computation takes “weeks”
  - Our result: 19 active S-boxes, found in 33.36 s
Comparison to other work

- Biryukov, Nikolić (EUROCRYPT 2010, SAC 2010)
  - determine byte differences, not just zero/non-zero difference
  - AES-192, 11 rounds:
    - Biryukov, Nikolić: 31 active S-boxes, computation takes "weeks"
    - Our result: 19 active S-boxes, found in 33.36 s
  - xAES-128, 10 rounds:
    - Nikolić: more than 22 active S-boxes ("a few hours on a single core") using split method
Comparison to other work

- Biryukov, Nikolić (EUROCRYPT 2010, SAC 2010)
  - determine byte differences, not just zero/non-zero difference
- AES-192, 11 rounds:
  - Biryukov, Nikolić: 31 active S-boxes, computation takes “weeks”
  - Our result: 19 active S-boxes, found in 33.36 s
- xAES-128, 10 rounds:
  - Nikolić: more than 22 active S-boxes (“a few hours on a single core”) using split method
  - Our result: 22 active S-boxes (2.68 s, single core) using split method
Comparison to other work

- Biryukov, Nikolić (EUROCRYPT 2010, SAC 2010)
  - determine byte differences, not just zero/non-zero difference
- AES-192, 11 rounds:
  - Biryukov, Nikolić: 31 active S-boxes, computation takes “weeks”
  - Our result: 19 active S-boxes, found in 33.36 s
- xAES-128, 10 rounds:
  - Nikolić: more than 22 active S-boxes (“a few hours on a single core”) using split method
  - Our result: 22 active S-boxes (2.68 s, single core) using split method
  - Not using split method: 25 active S-boxes (4 minutes on a single core, or 31.80 s using 24 cores)
How to program

- Complex bookkeeping of variables is unnecessary!
- Stay as close as possible to the original C code
How to program

- Complex bookkeeping of variables is unnecessary!
- Stay as close as possible to the original C code
  - `int next: index i for next unused x_i`
  - `int dummy: index j for next unused d_j`
How to program

- Complex bookkeeping of variables is unnecessary!
- Stay as close as possible to the original C code
  - `int next`: index $i$ for next unused $x_i$
  - `int dummy`: index $j$ for next unused $d_j$
  - remove round constants
How to program

- Complex bookkeeping of variables is unnecessary!
- Stay as close as possible to the original C code
  - `int next: index i for next unused x_i`
  - `int dummy: index j for next unused d_j`
  - remove round constants
  - intercept XOR, MixColumns: generate equations, increase `next` and `dummy`
How to program

- Complex bookkeeping of variables is unnecessary!
- Stay as close as possible to the original C code
  - `int next`: index $i$ for next unused $x_i$
  - `int dummy`: index $j$ for next unused $d_j$
  - remove round constants
  - intercept XOR, MixColumns: generate equations, increase `next` and `dummy`
  - intercept S-box: keep track of indices for objective function
How to program

- Complex bookkeeping of variables is unnecessary!
- Stay as close as possible to the original C code
  - `int next`: index $i$ for next unused $x_i$
  - `int dummy`: index $j$ for next unused $d_j$
  - remove round constants
  - intercept XOR, MixColumns: generate equations, increase `next` and `dummy`
  - intercept S-box: keep track of indices for objective function
  - More details: see source code
How to program

- Complex bookkeeping of variables is unnecessary!
- Stay as close as possible to the original C code
  - `int next`: index $i$ for next unused $x_i$
  - `int dummy`: index $j$ for next unused $d_j$
  - remove round constants
  - intercept XOR, MixColumns: generate equations, increase next and dummy
  - intercept S-box: keep track of indices for objective function
  - More details: see source code
- To debug/visualize: print indices $i$ of internal states $x_i$ and fill in solution
Related-key AES: How to program (2)

- Reference implementation (`rijndael-alg-ref.c`)
  - assume 256-bit key, 10 rounds

- Implementation needs $128 \cdot 11 = 1408$ key bits
  - but rounds up: $256 \cdot 6 = 1536$ key bits calculated

- Result: unnecessary S-box lookups, and wrong bounds!
  - solution: reorder loops and terminate sooner
Outline

1. Introduction

2. Three Easy, Automated Techniques
   - MILP Programming
   - SAT Solvers
   - Regular Expressions

3. Tools for Cryptography

4. Conclusion
SAT solvers: Introduction

- SAT solvers: input in Conjunctive Normal Form (CNF)
  - CNF = the ‘AND’ of a set of ‘OR’-clauses
  - Every variable = 1 bit
  - CryptoMiniSAT: also understands XOR clauses

- Example of CNF:
  
  \[
  (x_1 \lor \bar{x}_5 \lor x_4) \land \\
  (\bar{x}_1 \lor x_5 \lor x_3 \lor x_4) \land \\
  (\bar{x}_3 \lor \bar{x}_4)
  \]

- Conversion from C code to CNF needed
  - + convert back to interpret solution
SAT solvers: to CNF and back

- Custom approach: specific to certain (families of) ciphers
  - e.g. Grain of Salt (Soos): stream ciphers based on NLFSRs
SAT solvers: to CNF and back

- Custom approach: specific to certain (families of) ciphers
  - e.g. Grain of Salt (Soos): stream ciphers based on NLFSRs
- Using software to synthesize hardware circuits
  - e.g. CryptLogVer (Morawiecki et al.): Altera Quartus II + simple postprocessing
SAT solvers: to CNF and back

- Custom approach: specific to certain (families of) ciphers
  - e.g. Grain of Salt (Soos): stream ciphers based on NLFSRs
- Using software to synthesize hardware circuits
  - e.g. CryptLogVer (Morawiecki et al.): Altera Quartus II + simple postprocessing
- Tool to convert C code to CNF and back
  - e.g. **C32SAT** (Brummayer)
HAS-V Step Function
HAS-V Step Function: Local Collisions


A_t[0] → A_{t+1}[S] → A_{t+2}[0] → A_{t+3}[30] → A_{t+4}[30] → A_{t+5}[30]

B_{t+1}[0] → C_{t+2}[30] → D_{t+3}[30] → E_{t+4}[30]

50% 50% 50% (f_1, f_3) 50% 50% 50% (f_2)
HAS-V: Using C32SAT

- Chabaud and Joux: local collision = perturbation + correction(s)
  - Message words are reused: one message difference $W_i$ introduces many perturbations!
HAS-V: Using C32SAT

- Chabaud and Joux: local collision = perturbation + correction(s)
  - Message words are reused: one message difference $W_i$ introduces many perturbations!

- For HAS-V:
  - Step 0: Msg. diff. $W_0 = $ Perturb. $P_0$ (32-bit word)
  - Step 1: Msg. diff. $W_1 = $ Perturb. $P_1 \oplus $ Corr. ($P_0 \ll 11$)
HAS-V: Using C32SAT

- Chabaud and Joux: local collision = perturbation + correction(s)
  - Message words are reused: one message difference $W_i$ introduces many perturbations!

- For HAS-V:
  - Step 0: Msg. diff. $W_0 = \text{Perturb. } P_0$ (32-bit word)
  - Step 1: Msg. diff. $W_1 = \text{Perturb. } P_1 \oplus \text{Corr. } (P_0 \ll 11)$
  - Step 2: Msg. diff. $W_2 = \text{Perturb. } P_2 \oplus \text{Corr. } (P_1 \ll 7)$
    $\oplus \text{Corr. } (P_0 \land D_0)$

  - ... ($W_0, W_1, ...$ are reused in later steps)

- Dummy variable $D_0$: indicates if Boolean function $f$ absorbs or propagates difference
HAS-V: C32SAT results

- Processor: Intel Core 2 Duo E8400 @ 3GHz

- If $P_i \in \{00\ldots00_2, 11\ldots11_2\}$
  - Best solution: 192 local collisions for 60 steps (41s)
  - No solution with fewer than 192 local collisions (42s)
HAS-V: C32SAT results

- Processor: Intel Core 2 Duo E8400 @ 3GHz

  - If $P_i \in \{00..00_2, 11..11_2\}$
    - Best solution: 192 local collisions for 60 steps (41s)
    - No solution with fewer than 192 local collisions (42s)

  - If $P_i \in \{00..00_2, 01..01_2, 10..10_2, 11..11_2\}$
    - Best solution: 144 local collisions for 60 steps (3m 14s)
    - No solution with fewer than 144 local collisions (11m 10s)
HAS-V: C32SAT results

- Processor: Intel Core 2 Duo E8400 @ 3GHz

  - If $P_i \in \{00\ldots00_2, 11\ldots11_2\}$
    - Best solution: 192 local collisions for 60 steps (41s)
    - No solution with fewer than 192 local collisions (42s)

  - If $P_i \in \{00\ldots00_2, 01\ldots01_2, 10\ldots10_2, 11\ldots11_2\}$
    - Best solution: 144 local collisions for 60 steps (3m 14s)
    - No solution with fewer than 144 local collisions (11m 10s)

- More details: my Master’s thesis (only in Dutch)
Outline

1. Introduction

2. Three Easy, Automated Techniques
   - MILP Programming
   - SAT Solvers
   - Regular Expressions

3. Tools for Cryptography

4. Conclusion
Regular Expressions: Introduction

- Regular Expressions: to match patterns in strings
  - Text editor’s “Find” or ”Find/Replace”, but on steroids

- Included in several programming languages
  - Java, C++11, Apple’s Objective-C, C#, PHP, VB.NET, Python, Perl, JavaScript,...
Regular Expressions: Introduction

- Regular Expressions: to match patterns in strings
  - Text editor’s “Find” or ”Find/Replace”, but on steroids

- Included in several programming languages
  - Java, C++11, Apple’s Objective-C, C#, PHP, VB.NET, Python, Perl, JavaScript,...
## Regular Expressions: Notation

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x$</td>
<td>character $x$</td>
</tr>
<tr>
<td>.</td>
<td>any character</td>
</tr>
<tr>
<td>[ac]</td>
<td>character $a$ or $c$</td>
</tr>
<tr>
<td>[^a-c]</td>
<td>any character except $a$, $b$ or $c$ e.g. $d$, $A$, 9,…</td>
</tr>
<tr>
<td>([a-c])</td>
<td>save match for later use</td>
</tr>
<tr>
<td>$x^*$</td>
<td>zero or more $x$’s</td>
</tr>
<tr>
<td>$x+$</td>
<td>one or more $x$’s</td>
</tr>
<tr>
<td>$x?$</td>
<td>zero or one $x$’s</td>
</tr>
<tr>
<td>$x{m}$</td>
<td>exactly $m$ $x$’s</td>
</tr>
<tr>
<td>$x{m,}$</td>
<td>at least $m$ $x$’s</td>
</tr>
<tr>
<td>$x{m,n}$</td>
<td>at least $m$, but at most $n$ $x$’s</td>
</tr>
</tbody>
</table>
Meet-in-the-Middle Attack on XTEA

- XTEA: 64 rounds, 4 subkeys
  - One subkey used in every round
- Round key order as a string:
  03122130001322310010233201112033
  02112130031221310013223201102332
- Meet-in-the-middle attack on 23 rounds
  - First and last rounds: one subkey is not used
  - Middle rounds: all keys can be used, max. 15 rounds
  - Details: Sekar, Mouha, Velichkov, Preneel, CT-RSA 2010
- Regular expression?
Meet-in-the-Middle Attack on XTEA

- XTEA: 64 rounds, 4 subkeys
  - One subkey used in every round
- Round key order as a string:
  03122130001322310010233201112033
  02112130031221310013223201102332
- Meet-in-the-middle attack on 23 rounds
  - First and last rounds: one subkey is not used
  - Middle rounds: all keys can be used, max. 15 rounds
  - Details: Sekar, Mouha, Velichkov, Preneel, CT-RSA 2010
- \[^0]* \{1,15\}[^0]*,
Meet-in-the-Middle Attack on XTEA

- XTEA: 64 rounds, 4 subkeys
  - One subkey used in every round
- Round key order as a string:
  03122130001322310010233201112033
  02112130031221310013223201102332
- Meet-in-the-middle attack on 23 rounds
  - First and last rounds: one subkey is not used
  - Middle rounds: all keys can be used, max. 15 rounds
  - Details: Sekar, Mouha, Velichkov, Preneel, CT-RSA 2010

\[
\begin{align*}
\text{[^0]*.{1,15}[^0]*}, & \quad \text{[^1]*.{1,15}[^1]*}, \\
\text{[^2]*.{1,15}[^2]*}, & \quad \text{[^3]*.{1,15}[^3]*}
\end{align*}
\]
```perl
#!/usr/bin/perl

# XTEA key schedule
$x = "03122130001322310010233201112033" .
  "0211213031221310013223201102332";

# subkey $i is excluded in the outer rounds
for ($i=0; $i<4; $i++) {
  while ($x =~ /(^\$i)\{1,15\}(^\$i)/g) {
    if (length($1) >= 23) {  # show only 23-round attacks
      print length($1), "-round attack: ", $1,
      " (rounds: ", $-[1]+1, ", ", $+[1], ")\n"
    }
    pos($x) = $-[1]+1;  # matches may overlap
  }
}
```
Research papers should be verifiable
  ... releasing source code is therefore crucial!

ECRYPT II Tools for Cryptography
  [http://www.ecrypt.eu.org/tools](http://www.ecrypt.eu.org/tools)

Currently 16 tools listed
  Tools used in this lecture will be added today
  Other new submissions are very welcome!
Conclusion

- If we use an automated technique,
  - The execution time is unpredictable,
  - and the inner workings are not well understood.
Conclusion

- If we use an automated technique,
  - The execution time is unpredictable,
  - and the inner workings are not well understood.
- Yet, such techniques can be extremely useful:
  - typically not to break a cipher (requires too much time/memory)
  - but to use as a tool to construct an attack.
If we use an automated technique,
- The execution time is unpredictable,
- and the inner workings are not well understood.

Yet, such techniques can be extremely useful:
- typically not to break a cipher (requires too much time/memory)
- but to use as a tool to construct an attack.

How to do this? We gave examples for three approaches:
- MILP programming, SAT solvers, regular expressions.
Conclusion

- If we use an automated technique,
  - The execution time is unpredictable,
  - and the inner workings are not well understood.
- Yet, such techniques can be extremely useful:
  - typically not to break a cipher (requires too much time/memory)
  - but to use as a tool to construct an attack.
- How to do this? We gave examples for three approaches:
  - MILP programming, SAT solvers, regular expressions.

- Questions?