Privacy-preserving Security Protocols for RFIDs

Thesis defense

Mate Soos

INRIA Rhône-Alpes

6th of October 2009

Table of Contents

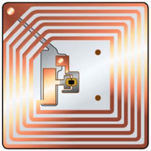
Outline

1. Context
   - RFID hardware
   - The privacy problem
   - Authentication in RFIDs
2. Ad-hoc protocols
   - ProbIP
   - EProbIP
3. Stream ciphers in RFIDs
   - Analysing stream ciphers with SAT solvers
   - Adapting SAT solvers to stream ciphers
   - Adapting stream cipher representation to SAT solvers
   - Attacks
4. Conclusions

What is an RFID?

An EPC RFID tag is:

- Small electronic device to identify items
- Projected to be on all items sold
- Cheap and disposable
- Used in the supply chain to track goods



RFID classification methods

By standards

- ISO 18000-*, 14443, 15693
- EPCglobal
- NFC

By frequencies

- Low Frequency (LF): 125/134.2 KHz
- High Frequency (HF): 13.56MHz (ISM)
- Ultra-HF (UHF): 856-930MHz
- Microwave Frequency: 2.4 GHz (ISM)

By power source

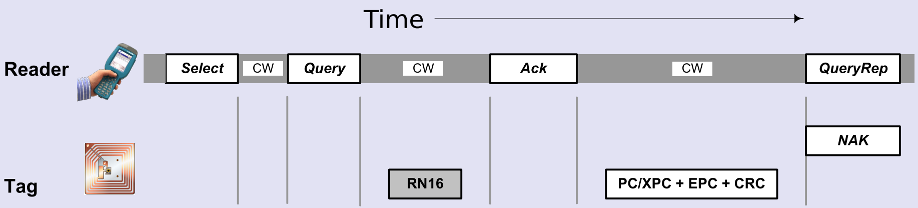
- Passive
- Semi-passive
- Active

The privacy problem

Causes

- RFIDs emit their ID to any query
- Their owners are easy to track
- Long read range, no line-of-sight
- Non human-detectable reader signal
- Unique ID

EPC protocol

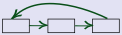
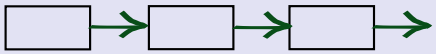


Solutions to the privacy problem

Physical layer-based

- Put the tag in a Faraday cage (ex.: mesh wallet)
- Kill the tag (ex.: EPC)
- Blocker tag, RFID Guardian
- Noisy tag
- Noisy reader

Protocol layer-based

- Pseudonym-rotation 
- Hash-based (ex.: OSK) 
- Keytree-based
- Ad-hoc primitives (ex. ProbIP)

Kill the tag

How it works

1. Give the tag a tag-specific 32-bit PIN code
2. The tag self-destructs

Advantages

- Easy to implement
- Once killed, cannot be re-awakened

Disadvantages

Loose many of RFIDs' advantages, e.g.:

- Automatic washing-machine
- Automatic recognition of items in the fridge
- Returning to shops defective items without receipts

Noisy tag

How it works

1. Generates pseudo-random noise on the channel
2. Sends reader the noise seed
3. Reader subtracts the noise and recovers the data

Advantages

- Simple to implement, should be cheap
- Perfect secrecy of data
- Multiple noisy tags enhance security

Disadvantages

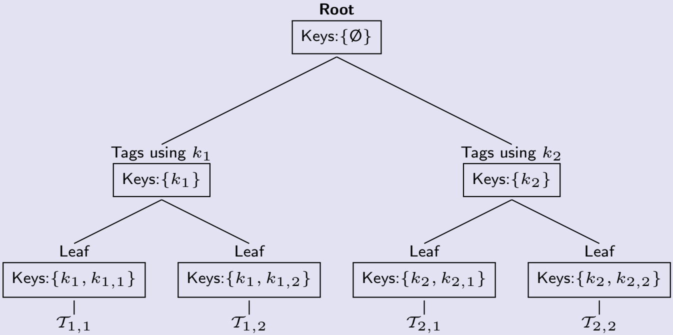
- Random noise needs to be known by the reader
- Needs to be worn all the time
- Implementation possibility has been questioned

Key-trees

Setup

- Tags are leaves of a multi-level tree
- Tag identifies itself with a key for each level
- Reader brute-forces each level
- This is $nlog_np$ speed, where n is depth, p is pop. size

Example

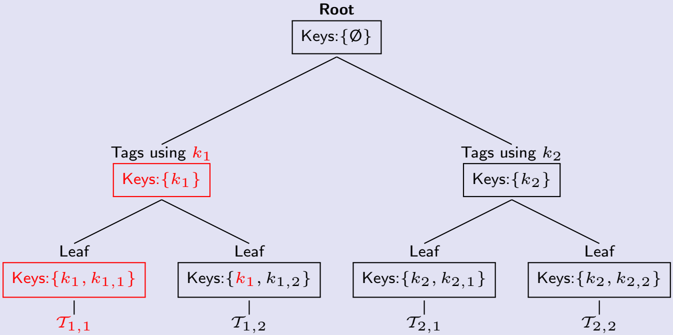


Key-trees

Setup

- Tags are leaves of a multi-level tree
- Tag identifies itself with a key for each level
- Reader brute-forces each level
- This is $n log_n p$ speed, where n is depth, p is pop. size

Example



Key-trees

Advantages

- Good privacy
- Fast (log-time identification)
- Extensively researched

Disadvantages

- Anonymity loss if tags are opened
- Needs cryptographic function

Authentication in RFIDs

What it is

- Used to verify that other party is who he claims to be
- Achieved through demonstration that secret is known

Why it is needed

- Against counterfeiting (e.g. medicines)
- Receiptless guarantee repairs

Solutions

- Challenge-response protocol using lightweight crypto-primitives (e.g. Grain)
- Physically Unclonable Functions (PUF)
- Rabin cryptosystem-based protocols
- LPN-based protocols (e.g. $HB^{\#}$)

Topic of the Thesis

RFIDs cannot use standard protocols

- Privacy protection
- Authentication service

RFIDs require

- Novel RFID protocols or crypto-primitives
- Analysis of these novel protocols for their security

Outline

Ad-hoc protocols — Motivations

- Standard ciphers seem not well-adapted to RFIDs
- By designing a protocol from scratch, it could better fit RFID constraints
- Could find potentially unexplored areas, and exploit them

ProbIP scheme

Public: keysize K, no. packets sent

Reader $\mathcal{R}$ | **Tag** $\mathcal{T}_i$

Database L:
$\{\ldots, (k_i, ID), \ldots\}$

Secret key: k_i

$\boxed{\text{HELLO} \longrightarrow}$

generate P packets
<a_1, b_1>,…,<a_L, b_L>
s.t. $a_j \in_r [1, K]$, $b_j \in_r \{0, 1\}$
$\sum_{j=1}^{L} \left[k_i[a_j] \oplus b_j \right] = L/2$

$\boxed{\longleftarrow \text{ generated packets}}$

find $(k_i, ID) \in L$
s.t. packets fit
$\sum_{j=1}^{L} \left[k_i[a_j] \oplus b_j \right] = L/2$

Breaking ProbIP

Ouafi et al. have broken the security of ProbIP.
Packets are represented as

$$\sum_{i=1}^{L} v_i^1 (K[i] \oplus b_i^1) = L/2$$

$$\sum_{i=1}^{L} v_i^2 (K[i] \oplus b_i^2) = L/2$$

$$\vdots$$

$$\sum_{i=1}^{L} v_i^l (K[i] \oplus b_i^l) = L/2$$

- l — no of packets gathered by the attacker
- v – indicator function of given key bit is in the packet

Resulting matrix is solved with Gaussian elimination, in poly-time

Error-introducing ProbIP

EProbIP is a extension to the original ProbIP protocol:

- Tags sometimes send erroneous packets
- Reader knows the possible key, so it can filter them
- Attacker cannot distinguish between packets

Setup:

1. Generate keys $(k_1, \ldots, k_n)$ uniquely and randomly with $\textsc{GenKey}$
2. Initialise $\mathcal{R}$ with keys $(k_1, \ldots, k_n)$
3. Set each $\mathcal{T}_i$'s key k_i with a $\textsc{SetKey}$ call

Phase 1 (Learning):

4. Let $\mathcal{A}$ do x_A $\textsc{TagInit}$ calls with $\mathcal{T}_A$ and records received packets into X_A
5. Let $\mathcal{A}$ do x_B $\textsc{TagInit}$ calls with $\mathcal{T}_B$ and records received packets into X_B

Phase 2 (Challenge):

6. $\mathcal{T}_C \stackrel{r}{\leftarrow} \{\mathcal{T}_A, \mathcal{T}_B\}$
7. $\mathcal{A}$ performs x_C $\textsc{TagInit}$ calls with $\mathcal{T}_C$ and records received packets into X_C
8. $\mathcal{A}$ performs calculations on the recorded packets to guess $\mathcal{T}_C \stackrel{?}{=} \mathcal{T}_A$

Experiment succeeds if $\mathcal{A}$ guessed $\mathcal{T}_C$ correctly

How can the attacker win the privacy exp.?

Possible methods

1. Find a key that fits most packets — using a MaxSAT solver
2. Use a tailor-made approach using out that the error rate is low

How can the attacker win the privacy exp.?

Possible methods

1. Find a key that fits most packets — using a MaxSAT solver
2. Use a tailor-made approach using out that the error rate is low

1) Using MaxSAT solvers

- Solves for *any* error rate
- Can work on a small amount of packets
- Does not benefit from more packets

How can the attacker win the privacy exp.?

Possible methods

1. Find a key that fits most packets — using a MaxSAT solver
2. Use a tailor-made approach using out that the error rate is low

1) Using MaxSAT solvers

- Solves for *any* error rate
- Can work on a small amount of packets
- Does not benefit from more packets

2) Using strategy adapted to low error rate

- Needs a large amount of packets to work
- Can benefit from low error rate
- Benefits from more packets

Strategy adapted to low error-rate

Input: packets $X_A \cup X_C$
Output: $\mathcal{T}_A = \mathcal{T}_C$ or $\mathcal{T}_A \neq \mathcal{T}_C$

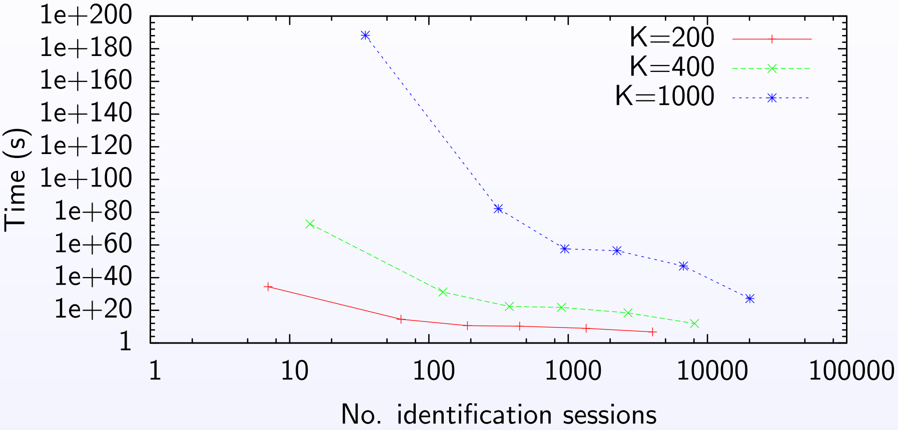
Pick a set of k most prevalent key bits;
foreach *combination of* `true-false` *for the picked bits* **do**
    picked key bits $\leftarrow$ selected combination;
    while *enough packets indicate: key bit must be set to a value* **do**
        key bit $\leftarrow$ value indicated;
    end
    if *all key bits are set and the satisfied portion of packets is about* $1 - err$ **then**
        return $\mathcal{T}_A = \mathcal{T}_C$;
    end
end
return $\mathcal{T}_A \neq \mathcal{T}_C$;

Implementation: in MiniSat

Modified MiniSat such that:

- Inferences are made based on multiple packets
- X number of packets needed to make an inference
- The X the larger, the more 'robust' the solving
- But more information will be lost
- i.e. more packets $\rightarrow$ faster solving

Security rating results



Ad-hoc protocols — What have we learnt

- Ad-hoc primitives need multiple cycles of design&analysis
- Difficult to evaluate the security of the resulting schemes
- Can take many years to develop a robust ad-hoc protocol

Outline

Stream ciphers in RFIDs

Motivations

- We have seen that ad-hoc protocols are notoriously un-robust
- Stream ciphers could be adapted to RFIDs — eSTREAM project
- Analysis of hardware-oriented stream ciphers is possible with SAT solvers

Contributions

- Adapt the SAT solver to the environment of cryptography
- Adapt the stream cipher's representation to SAT solvers
- Solve a number of ciphers

What is a SAT solver

Solves a problem in CNF

CNF is an "and of or-s"

$$\neg x_1 \vee \neg x_3 \qquad \neg x_2 \vee x_3 \qquad x_1 \vee x_2$$

Uses DPLL(φ) algorithm

1. If formula φ is trivial, return SAT/UNSAT
2. Picks a variable v to branch on
3. $v \leftarrow$ value
4. Simplifies formula to φ' and calls DPLL(φ')
5. if SAT, output SAT
6. if UNSAT, $v \leftarrow$ opposite value
7. Simplifies formula to φ'' and calls DPLL(φ'')
8. if SAT, output SAT
9. if UNSAT, output UNSAT

Problem with XOR-s

The truth

$$a \oplus b \oplus c$$

must be put into the solver as

$$a \vee \overline{b} \vee \overline{c} \qquad (1) \qquad\qquad \overline{a} \vee \overline{b} \vee c \qquad (2)$$
$$a \vee b \vee c \qquad (3) \qquad\qquad \overline{a} \vee b \vee \overline{c} \qquad (4)$$

So, straightforward conversion takes 2^{n-1} clauses to model an n-long XOR

Solution until now

Example

$$x_1 \oplus x_2 \oplus x_3 \oplus x_4 \oplus x_5 \oplus x_6 \oplus x_7 \oplus x_8 \oplus x_9$$

Modelled in CNF:

$$\neg i_1 \oplus x_1 \oplus x_2 \oplus x_3 \oplus x_4$$
$$\neg i_2 \oplus x_5 \oplus x_6 \oplus x_7 \oplus x_8 \oplus x_9$$
$$i_1 \oplus i_2$$

Problems

- Still very long to model
- Needs extra vars

Solution until now

Example

$$x_1 \oplus x_2 \oplus x_3 \oplus x_4 \oplus x_5 \oplus x_6 \oplus x_7 \oplus x_8 \oplus x_9$$

Modelled in CNF:

$$\neg i_1 \oplus x_1 \oplus x_2 \oplus x_3$$
$$\neg i_2 \oplus x_4 \oplus x_5 \oplus x_6$$
$$\neg i_3 \oplus x_7 \oplus x_8 \oplus x_9$$
$$i_1 \oplus i_2 \oplus i_3$$

Problems

- Still very long to model
- Needs extra vars

Solution to XOR: xor-clause

Example

$$a \oplus b \oplus c$$

Represents regular clauses

$$a \vee \neg b \vee \neg c \qquad (1) \qquad \neg a \vee \neg b \vee c \qquad (2)$$
$$a \vee b \vee c \qquad (3) \qquad \neg a \vee b \vee \neg c \qquad (4)$$

changes appearance to match the situation

Example set-up

$$a = \texttt{true} \quad b = \texttt{true} \quad c = \texttt{false}$$

$$\Rightarrow \neg a \vee \neg b \vee c$$

Solution to XOR: xor-clause

Example

$$a \oplus b \oplus c$$

Represents regular clauses

$$a \vee \neg b \vee \neg c \qquad (1) \qquad\qquad \neg a \vee \neg b \vee c \qquad (2)$$
$$a \vee b \vee c \qquad (3) \qquad\qquad \neg a \vee b \vee \neg c \qquad (4)$$

changes appearance to match the situation

Results

- 2.2x speed
- Order of magnitude savings in memory

Solution to XOR: xor-clause

Example

$$a \oplus b \oplus c$$

Represents regular clauses

$$a \vee \neg b \vee \neg c \quad (1) \qquad \neg a \vee \neg b \vee c \quad (2)$$
$$a \vee b \vee c \quad (3) \qquad \neg a \vee b \vee \neg c \quad (4)$$

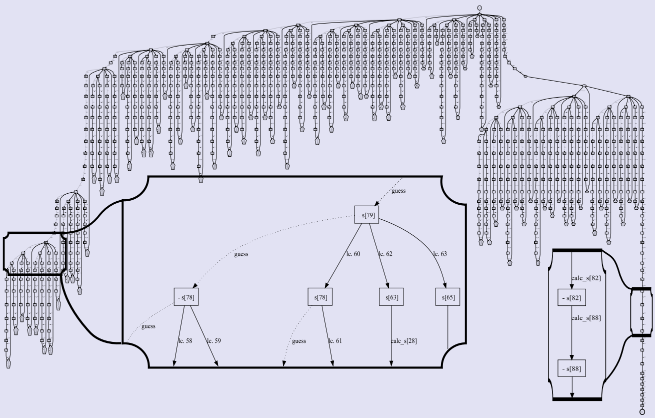
changes appearance to match the situation

Challenges overcome

- MiniSat is complex, we needed to completely understand it
- Design choices were difficult: e.g. we use special memory alloc. to maximise cache-hit

Dynamic behaviour analysis

Example search tree



Visualised

- Guesses
- Propagations
- Generated learnt clauses
- Clause group causing the propagation

Calculated stats

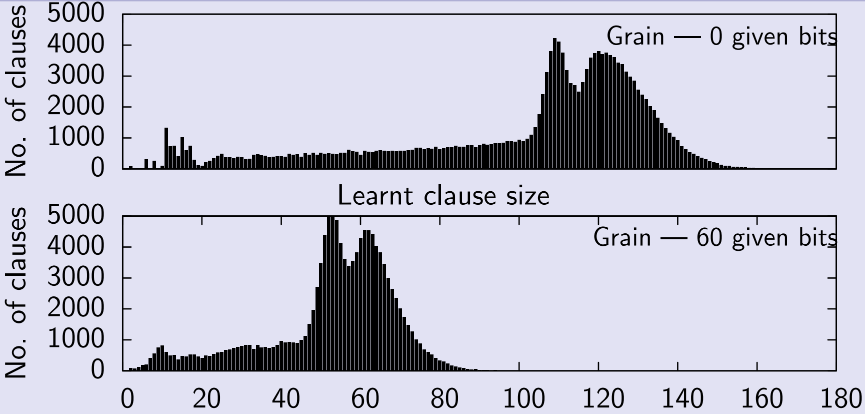
- Average depth
- Most conflicted clauses
- No. of guess/branch
- Most guessed vars
- Most propagated vars

Statistics generated

Further stats

- Learnt clause size distribution
- Branch length distribution

Ex. learnt clause distribution



Gaussian elimination

Reasoning

- Gaussian elimination is efficient for solving systems of linear equations
- xor-clause is a linear equation $\rightarrow$ use Gauss elim. to solve them

Implementation

A-matrix

$$\begin{array}{ccccc} v10 & v8 & v9 & v12 & \text{aug} \end{array}$$
$$\left[\begin{array}{cccc|c} 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 \end{array}\right]$$

N-matrix

$$\begin{array}{ccccc} v10 & v8 & v9 & v12 & \text{aug} \end{array}$$
$$\left[\begin{array}{cccc|c} 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 \end{array}\right]$$

Gaussian elimination

Reasoning

- Gaussian elimination is efficient for solving systems of linear equations
- xor-clause is a linear equation $\rightarrow$ use Gauss elim. to solve them

Implementation

A-matrix
with $v8$ assigned to `true`

$$\begin{array}{ccccc} v10 & v8 & v9 & v12 & \text{aug} \end{array}$$
$$\left[\begin{array}{cccc|c} 1 & - & 1 & 1 & 1 \\ 0 & - & 1 & 1 & 1 \\ 0 & - & 0 & 1 & 0 \\ 0 & - & 0 & 0 & 0 \end{array}\right]$$

N-matrix

$$\begin{array}{ccccc} v10 & v8 & v9 & v12 & \text{aug} \end{array}$$
$$\left[\begin{array}{cccc|c} 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 \end{array}\right]$$

Gaussian elimination

Reasoning

- Gaussian elimination is efficient for solving systems of linear equations
- xor-clause is a linear equation $\rightarrow$ use Gauss elim. to solve them

Implementation

A-matrix
with $v8$ assigned to `true`

$$\begin{array}{ccccc} v10 & v8 & v9 & v12 & \text{aug} \end{array}$$
$$\left[\begin{array}{cccc|c} 1 & - & 1 & 1 & 1 \\ 0 & - & 1 & 1 & 1 \\ 0 & - & 0 & 1 & 0 \\ 0 & - & 0 & 0 & 0 \end{array}\right]$$

N-matrix

$$\begin{array}{ccccc} v10 & v8 & v9 & v12 & \text{aug} \end{array}$$
$$\left[\begin{array}{cccc|c} 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 \end{array}\right]$$

Resulting xor-clause:

$$v8 \oplus v12$$

Gaussian elimination

Reasoning

- Gaussian elimination is efficient for solving systems of linear equations
- xor-clause is a linear equation $\rightarrow$ use Gauss elim. to solve them

Implementation

A-matrix
with $v8$ assigned to `true`

$$\begin{array}{ccccc} v10 & v8 & v9 & v12 & \text{aug} \end{array}$$
$$\left[\begin{array}{cccc|c} 1 & - & 1 & 1 & 1 \\ 0 & - & 1 & 1 & 1 \\ 0 & - & 0 & 1 & 0 \\ 0 & - & 0 & 0 & 0 \end{array}\right]$$

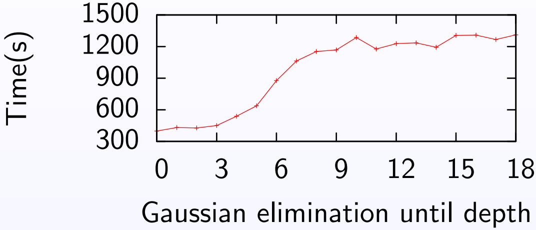
N-matrix

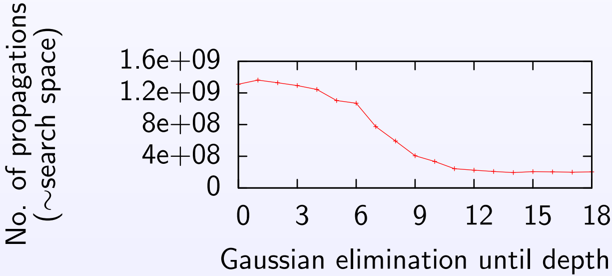
$$\begin{array}{ccccc} v10 & v8 & v9 & v12 & \text{aug} \end{array}$$
$$\left[\begin{array}{cccc|c} 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 \end{array}\right]$$

Resulting xor-clause:

$$v12 = \texttt{false} \quad \leftarrow \quad v8 \oplus v12$$

Gaussian elimination results





Gaussian elimination results

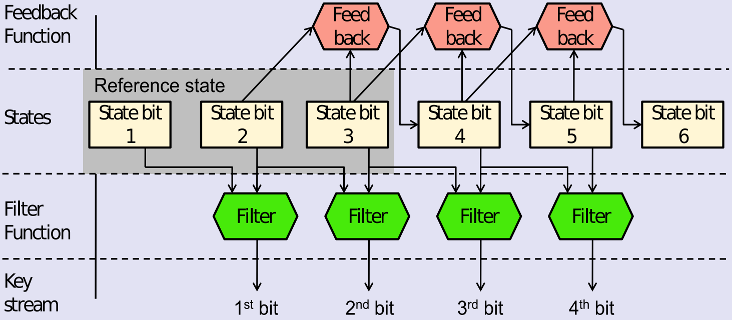
	No. help bits	Gaussian elimination active until level		
		Inactive	2	3
Crypto-1	12	27.0 s	25.8 s(4%)	26.5 s(2%)
HiTag2	18	34.8 s	33.9 s(3%)	29.5 s(15%)
Bivium B	60	174.0 s	165.1 s(5%)	171.1 s(2%)

Highlights

- Search space reduced by up to 87%
- Speedup between 0-15%
- A mix of linear and non-linear methods
- Adds possibility to add other algebraic tools → potentially major speedup

Logical circuit representation

Example



Legend

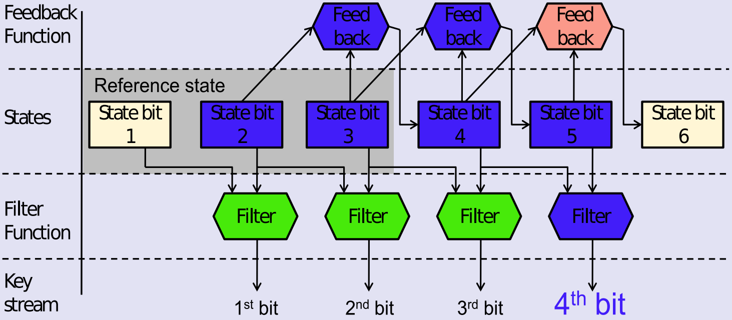
- Variables → boxes
- Functions → hexagons

Complexity measures

- *Depth* of keystream bit
- *Dependency no.*: state ↔ keystream
- *Difficulty of functions*: representation

Logical circuit representation

Example



Legend

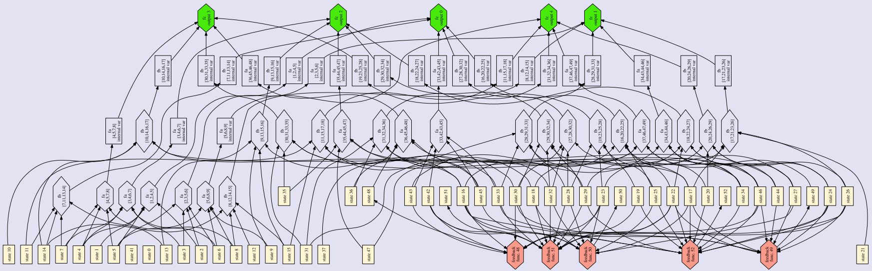
- Variables → boxes
- Functions → hexagons

Complexity measures

- *Depth* of keystream bit
- *Dependency no.*: state ↔ keystream
- *Difficulty of functions*: representation

Dependency graph generator

Example HiTag2 logical circuit

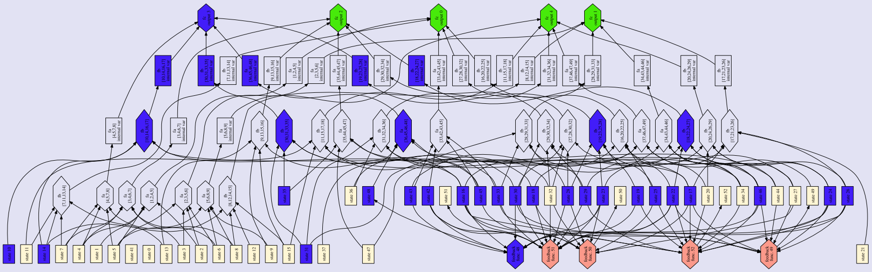


Usage

- Calculate mentioned statistics
- Visual clue

Dependency graph generator

Example HiTag2 logical circuit



Usage

- Calculate mentioned statistics
- Visual clue

Optimising representation of non-linear functions

Example $\mathbb{GF}(2)$ polynomial

$$x_1 + x_1x_2 + x_2x_3 + x_1x_3$$

Usual representation

$$x_1 + i_1 + i_2 + i_3$$

- No. of clauses: 3×3 regular $+$ 1 xor-clause
- $\sum$ clause length: 31
- 2 extra variables

Karnaugh-table representation

$$\neg x_1 \vee \neg x_3 \quad \neg x_2 \vee x_3 \quad x_1 \vee x_2$$

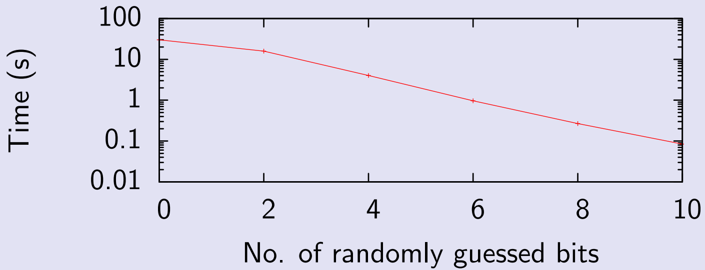
- No. of clauses: 3 regular
- $\sum$ clause length: 6
- No extra variables

Crypto-1

Background

- Used for micropayment in public transport
- Best SAT solver-based attack : 200 s to solve on avg.
- Best non-SAT solver-based attack: 0.1 s through algebraic attack

Our techniques



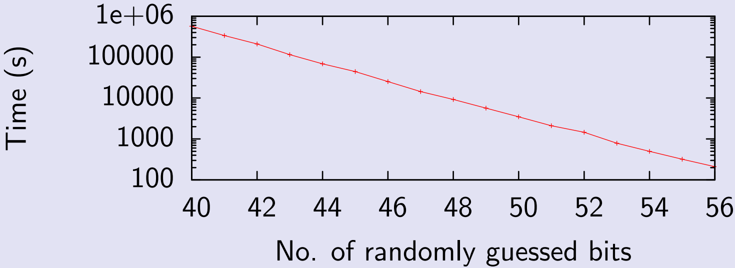
Find its secret state in approx. 40 s

Bivium B

Background

- Simplified version of Trivium eSTREAM candidate
- Best SAT solver-based attack against it takes 2^{43} s
- Non-SAT solver-based attack: $2^{64.5}$ s

Our techniques



Find its secret state in approx. $2^{36.5}$ s

Stream ciphers in RFIDs — What we have learnt

- SAT solvers are useful to study hardware-oriented stream ciphers
- Best results are achieved by adapting both solvers to ciphers and cipher's representation to solvers
- Such a system is able to break certain ciphers

Outline

Contributions of the Thesis

Contributions

- Created an in-depth state of the art
- Conceived two ad-hoc protocols, ProbIP [1] and EProbIP
- Analysed the Di Pietro-Molva ad-hoc protocol [2]
- Improved SAT solver-based cryptographic attacks [3,4]

References

1. "Secret Shuffling: A Novel Approach to RFID Private Identification" by Castelluccia and Soos, RFIDSec'07
2. "Analysing the Molva and Di Pietro Private RFID Authentication Scheme" by Soos, RFIDSec'08
3. "Solving Low-Complexity Ciphers with Optimized SAT solvers" by Nohl and Soos, EUROCRYPT'09 (poster)
4. "Extending SAT Solvers to Cryptographic Problems" by Soos, Castelluccia and Nohl, SAT'09

Conclusions

- RFID hardware is unnatural to optimise for
- Ad-hoc protocols are notoriously fragile, but could be a solution in the long run
- For immediate use, standard crypto-primitives optimised for RFIDs (e.g. HW-oriented stream ciphers) are more suited

Future work

- Post-Doc in the SALSA team of INRIA
- Distributed SAT solving
- Iterative SAT solving
- Mix of SAT solving and algebraic techniques
- RFID-AP ANR project

Thank you for your time

Di Pietro-Molva scheme

The Di-Pietro Molva scheme works as follows:

1. Tag generates nonces $r_1 \ldots r_2$
2. Tag sends $\alpha_p = r_p \oplus k$
3. Tag sends $V[p] = \mathsf{DPM}(r_p)$
4. Reader computes $\mathsf{DPM}(\alpha_p \oplus k) = V'[p]$ for all k — the one that fits is the tag
5. Once tag is identified, authentication takes place

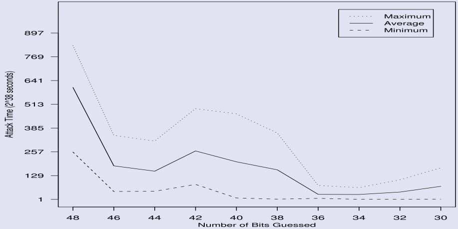
Found shortcomings

Problems found in the scheme (published as):

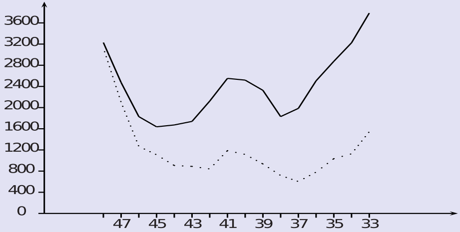
- Does not scale — finding tag is linear in population size
- Due to func. DPM, there are $2^{2|k|/3}$ key-equivalence classes (i.e. identification is bad)
- $(\alpha_p, V[p])$ pairs do not always contain enough information (pairs are not independent)
- DPM is not secure, each protocol run reveals 1 bit of secret key

Research results until now

"Attacking Bivium with MiniSat" by (McDonald et al.)



"Attacking Bivium Using SAT Solvers" by (Eibach et al.)



Research results until now

We introduce more randomness

- Reference state bits to assign are picked randomly
- The picked bits are assigned randomly `true` or `false`
- Clauses are randomly permutated inside MiniSat
- MiniSat's internal seed (used to randomly explore the search space) is set randomly
- MiniSat's random number generator has been replaced

How it works (ex. $\text{Random-HB}^{\#}$)

Reader $\mathcal{R}$		**Tag** $\mathcal{T}_i$
Secrets X, Y		Secrets X, Y
		$\nu \in_R \{\{0,1\}^m \mid$ Prob.$(\nu_i = 1) = \eta$ for $1 \leq i \leq m\}$
		Choose $\mathbf{b} \in_R \{0,1\}^{k_Y}$
	$\boxed{\mathbf{b} \longleftarrow}$	
Choose $\mathbf{a} \in_R \{a,1\}^{k_X}$		
	$\boxed{\longrightarrow \mathbf{a}}$	
		Let $\mathbf{z} = \mathbf{a} \cdot C \oplus \mathbf{b} \cdot Y \oplus \nu$
	$\boxed{\mathbf{z} \longleftarrow}$	
Check Hwt$(\mathbf{a} \cdot X \oplus \mathbf{b} \cdot Y \oplus \mathbf{z}) \leq um$		

HB# cont.

Advantages

- Simple to implement: needs XOR, random number generator
- Protocol is well-analysed by its authors

Disadvantages

- Transferred data is large ($\rightarrow$ slow)
- LPN problem quite unresearched, new research is pushing up secure parameter sizes

Example protocol No. 1

Reader $\mathcal{R}_j$		**Tag** $\mathcal{T}_i$
Generate nonce IV_1		
	$\longrightarrow IV_1$	
		Generate nonce IV_2 and calculate $\sigma = ID \oplus cipher(k, IV_1 \oplus IV_2)$
	$\longleftarrow IV_2, \sigma$	
find $(k, ID) \in L$ s.t. $ID = \sigma \oplus cipher(k, IV_1 \oplus IV_2)$		

Example protocol No. 2

Reader $\mathcal{R}_j$		**Tag** $\mathcal{T}_i$
Generate nonce IV_1		
	$\longrightarrow IV_1$	
		Generate nonce IV_2 and calculate $M = cipher(IV_1, IV_2)$ $\sigma = ID \oplus cipher(k, M)$
	$\longleftarrow IV_2, \sigma$	
calculate $M = cipher(IV_1, IV_2)$ find $(k, ID) \in L$ s.t. $ID = \sigma \oplus cipher(k, M)$		
	optional — only for mutual authentication	
calculate $\tau = ID \oplus cipher(k, M \oplus 1)$		
	$\longrightarrow \tau$	
		check $\tau \stackrel{?}{=} ID \oplus cipher(k, M \oplus 1)$