# Cryptanalysis of The Atmel Cipher in SecureMemory, CryptoMemory and CryptoRF

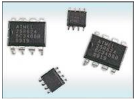
Alex Biryukov, Ilya Kizhvatov and Bin Zhang

Laboratory of Algorithmics, Security and Cryptology
University of Luxembourg

7-June-2011

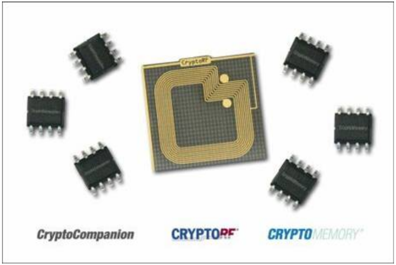
# Outline

1. Atmel Product Family
2. The Cipher and the Environment
3. Our Attack
4. Practical Implementation
5. Conclusions




ATMEL
Secure Silicon Innovations
CRYPTOMEMORY

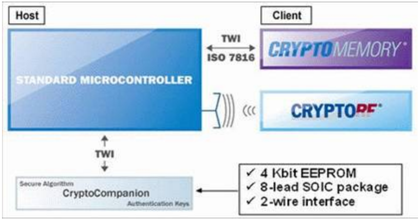

ATMEL
CRYPTORF™
AT88SC0404CRF
CryptoRF 4K Bits in 4 User Zones
CryptoRF@atmel.com
MAKING
SMART CARDS SMARTER™


CryptoCompanion
CRYPTORF
CRYPTOMEMORY


ATMEL
CRYPTORF
AT88SC0808CRF
MAKING

# Atmel Product Family AT88SC

- SecureMemory (SM, 1999)
- CryptoMemory (CM, 2002, Successor of SM)
- SM and CM are ISO/IEC 7816 smart cards
- CryptoRF (CR) = CryptoMemory + RF module (2003)
- CR is ISO/IEC 14443-B smart card



# Applications

- ID and access cards
- healthcare
- loyalty cards
- e-purses
- energy meters
- e-government
- printers and print cartridges
- Digital-TV
- subassembly authentication
- ...



# The Atmel Cipher

- A proprietary stream cipher which has 2 versions.
- The simple version is used in SM, while the more complex version is adopted in CM and CR.
- There is 1 byte feedback of the output into the other 3 shift registers in the complex version.
- It is commonly believed that the complex version provides much stronger security.

# The Atmel Cipher

- A proprietary stream cipher which has 2 versions.
- The simple version is used in SM, while the more complex version is adopted in CM and CR.
- There is 1 byte feedback of the output into the other 3 shift registers in the complex version.
- It is commonly believed that the complex version provides much stronger security.

# The Atmel Cipher

- A proprietary stream cipher which has 2 versions.
- The simple version is used in SM, while the more complex version is adopted in CM and CR.
- There is 1 byte feedback of the output into the other 3 shift registers in the complex version.
- It is commonly believed that the complex version provides much stronger security.

# The Atmel Cipher

- A proprietary stream cipher which has 2 versions.
- The simple version is used in SM, while the more complex version is adopted in CM and CR.
- There is 1 byte feedback of the output into the other 3 shift registers in the complex version.
- It is commonly believed that the complex version provides much stronger security.

# The Atmel Cipher

- A proprietary stream cipher which has 2 versions.
- The simple version is used in SM, while the more complex version is adopted in CM and CR.
- There is 1 byte feedback of the output into the other 3 shift registers in the complex version.
- It is commonly believed that the complex version provides much stronger security.

# The Atmel Ciphers

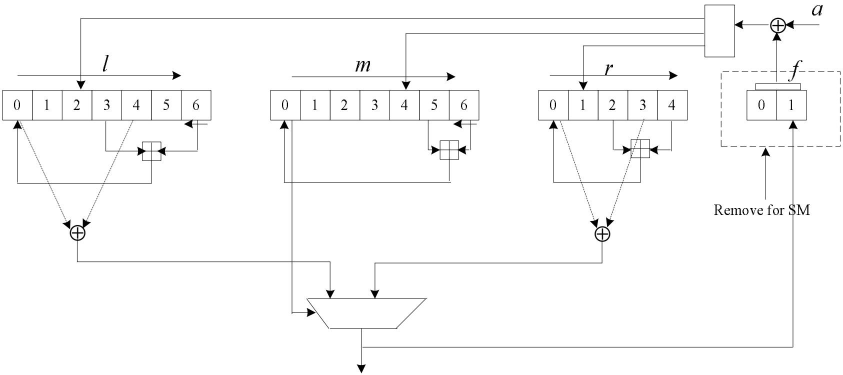



Figure: The Atmel ciphers

# How It Works

At each tick, a cipher state $s = (l, m, r, f) \in \mathbb{F}_2^{117}$ (for SM, ignore $f$ and $s \in \mathbb{F}_2^{109}$) is converted into a successor state $s' = (l', m', r', f')$ as follows.

- Inject the input $a$ into $s$ at several cell positions, resulting in an intermediate state $\hat{s}$. For CM, let $b = a \oplus f_0 f_1$; while for SM, let $b = a$.
- Shift the left, right and middle registers one cell to the right and compute the new 0th terms by the 1-bit left rotation $L$ and the modified modular addition $\boxplus$.

$$l'_{i+1} := \hat{l}_i, \;\; m'_{i+1} := \hat{m}_i, \;\; \text{for} \;\; i \in \{0, 1, \ldots, 5\},$$
$$r'_{i+1} := \hat{r}_i \;\; \text{for} \;\; i \in \{0, 1, \ldots, 3\},$$
$$l'_0 := \hat{l}_3 \boxplus L(\hat{l}_6), \;\; m'_0 := \hat{m}_5 \boxplus L(\hat{m}_6), \;\; r'_0 := \hat{r}_2 \boxplus \hat{r}_4.$$

# How It Works

At each tick, a cipher state $s = (l, m, r, f) \in \mathbb{F}_2^{117}$ (for SM, ignore $f$ and $s \in \mathbb{F}_2^{109}$) is converted into a successor state $s' = (l', m', r', f')$ as follows.

- Inject the input $a$ into $s$ at several cell positions, resulting in an intermediate state $\hat{s}$. For CM, let $b = a \oplus f_0 f_1$; while for SM, let $b = a$.
- Shift the left, right and middle registers one cell to the right and compute the new 0th terms by the 1-bit left rotation $L$ and the modified modular addition $\boxplus$.

$$
\begin{aligned}
&l'_{i+1} := \hat{l}_i, \;\; m'_{i+1} := \hat{m}_i, \;\; \text{for} \;\; i \in \{0, 1, \ldots, 5\},\\
&r'_{i+1} := \hat{r}_i \;\; \text{for} \;\; i \in \{0, 1, \ldots, 3\},\\
&l'_0 := \hat{l}_3 \boxplus L(\hat{l}_6), \;\; m'_0 := \hat{m}_5 \boxplus L(\hat{m}_6), \;\; r'_0 := \hat{r}_2 \boxplus \hat{r}_4.
\end{aligned}
$$

# How It Works

At each tick, a cipher state $s = (l, m, r, f) \in \mathbb{F}_2^{117}$ (for SM, ignore $f$ and $s \in \mathbb{F}_2^{109}$) is converted into a successor state $s' = (l', m', r', f')$ as follows.

- Inject the input $a$ into $s$ at several cell positions, resulting in an intermediate state $\hat{s}$. For CM, let $b = a \oplus f_0 f_1$; while for SM, let $b = a$.
- Shift the left, right and middle registers one cell to the right and compute the new 0th terms by the 1-bit left rotation $L$ and the modified modular addition $\boxplus$.

$$
\begin{aligned}
&l'_{i+1} := \hat{l}_i, \;\; m'_{i+1} := \hat{m}_i, \;\; \text{for} \;\; i \in \{0, 1, \dots, 5\}, \\
&r'_{i+1} := \hat{r}_i \;\; \text{for} \;\; i \in \{0, 1, \dots, 3\}, \\
&l'_0 := \hat{l}_3 \boxplus L(\hat{l}_6), \;\; m'_0 := \hat{m}_5 \boxplus L(\hat{m}_6), \;\; r'_0 := \hat{r}_2 \boxplus \hat{r}_4.
\end{aligned}
$$

# Cont.

- Generate the keystream and shift the feedback register $f$ one cell to the left and set a new 1st entry as the output nibble for CM.
- Let $outputl(l') = l'_{0,1} \oplus l'_{4,1} \parallel l'_{0,2} \oplus l'_{4,2} \parallel l'_{0,3} \oplus l'_{4,3} \parallel l'_{0,4} \oplus l'_{4,4}$ the rightmost 4 bits of $l'_0 \oplus l'_4$, and $outputr(r') = r'_{0,1} \oplus r'_{3,1} \parallel r'_{0,2} \oplus r'_{3,2} \parallel r'_{0,3} \oplus r'_{3,3} \parallel r'_{0,4} \oplus r'_{3,4}$ the rightmost 4 bits of $r'_0 \oplus r'_3$.
- The output of $s'$, denoted by $output(s')$, is given by

$$output(s')_i = \begin{cases} outputl(l')_i, & \text{if } m'_{0,i+3} = 0 \\ outputr(r')_i, & \text{if } m'_{0,i+3} = 1. \end{cases} \quad i \in \{0, \ldots, 3\}.$$

# Cont.

- Generate the keystream and shift the feedback register $f$ one cell to the left and set a new 1st entry as the output nibble for CM.
- Let $outputl(l') = l'_{0,1} \oplus l'_{4,1} \parallel l'_{0,2} \oplus l'_{4,2} \parallel l'_{0,3} \oplus l'_{4,3} \parallel l'_{0,4} \oplus l'_{4,4}$ the rightmost 4 bits of $l'_0 \oplus l'_4$, and $outputr(r') = r'_{0,1} \oplus r'_{3,1} \parallel r'_{0,2} \oplus r'_{3,2} \parallel r'_{0,3} \oplus r'_{3,3} \parallel r'_{0,4} \oplus r'_{3,4}$ the rightmost 4 bits of $r'_0 \oplus r'_3$.
- The output of $s'$, denoted by $output(s')$, is given by

$$output(s')_i = \begin{cases} outputl(l')_i, & \text{if } m'_{0,i+3} = 0 \\ outputr(r')_i, & \text{if } m'_{0,i+3} = 1. \end{cases} \quad i \in \{0, \ldots, 3\}.$$

# Cont.

- Generate the keystream and shift the feedback register $f$ one cell to the left and set a new 1st entry as the output nibble for CM.

- Let $outputl(l') = l'_{0,1} \oplus l'_{4,1} \parallel l'_{0,2} \oplus l'_{4,2} \parallel l'_{0,3} \oplus l'_{4,3} \parallel l'_{0,4} \oplus l'_{4,4}$ the rightmost 4 bits of $l'_0 \oplus l'_4$, and
$outputr(r') = r'_{0,1} \oplus r'_{3,1} \parallel r'_{0,2} \oplus r'_{3,2} \parallel r'_{0,3} \oplus r'_{3,3} \parallel r'_{0,4} \oplus r'_{3,4}$ the rightmost 4 bits of $r'_0 \oplus r'_3$.

- The output of $s'$, denoted by $output(s')$, is given by

$$output(s')_i = \begin{cases} outputl(l')_i, & \text{if } m'_{0,i+3} = 0 \\ outputr(r')_i, & \text{if } m'_{0,i+3} = 1. \end{cases} \quad i \in \{0, \ldots, 3\}.$$

# Cont.

- Generate the keystream and shift the feedback register $f$ one cell to the left and set a new 1st entry as the output nibble for CM.
- Let $outputl(l') = l'_{0,1} \oplus l'_{4,1} \parallel l'_{0,2} \oplus l'_{4,2} \parallel l'_{0,3} \oplus l'_{4,3} \parallel l'_{0,4} \oplus l'_{4,4}$ the rightmost 4 bits of $l'_0 \oplus l'_4$, and $outputr(r') = r'_{0,1} \oplus r'_{3,1} \parallel r'_{0,2} \oplus r'_{3,2} \parallel r'_{0,3} \oplus r'_{3,3} \parallel r'_{0,4} \oplus r'_{3,4}$ the rightmost 4 bits of $r'_0 \oplus r'_3$.
- The output of $s'$, denoted by $output(s')$, is given by

$$output(s')_i = \begin{cases} outputl(l')_i, & \text{if } m'_{0,i+3} = 0 \\ outputr(r')_i, & \text{if } m'_{0,i+3} = 1. \end{cases} \quad i \in \{0, \ldots, 3\}.$$

# The Authentication Protocol

In the protocol, the tag and the reader exchange the nonces and use the cipher to generate keystream that will be used as authenticators for both sides.

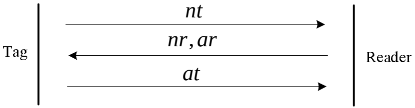



Figure: The authentication protocol

# Initialization

- Let $nt \in (\mathbb{F}_2^8)^8$ be a tag nonce, $nr \in (\mathbb{F}_2^8)^8$ a reader nonce and $k \in (\mathbb{F}_2^8)^8$ be the shared key between the tag and the reader.
- Initialize the registers $l$, $m$, $r$ and $f$ (for SM, ignore $f$) to be zero.
- Clock the cipher as follows.

$$
\begin{aligned}
s_0 &:= 0, \\
s_{i+1} &:= suc(nr_i, suc^v(nt_{2i+1}, suc^v(nt_{2i}, s_i))), && i \in \{0, \dots, 3\} \\
s_{i+5} &:= suc(nr_{i+4}, suc^v(k_{2i+1}, suc^v(k_{2i}, s_{i+4}))), && i \in \{0, \dots, 3\}
\end{aligned}
$$

where $v = 1$ for SM and $v = 3$ for CM.

# Initialization

- Let $nt \in (\mathbb{F}_2^8)^8$ be a tag nonce, $nr \in (\mathbb{F}_2^8)^8$ a reader nonce and $k \in (\mathbb{F}_2^8)^8$ be the shared key between the tag and the reader.
- Initialize the registers $l$, $m$, $r$ and $f$ (for SM, ignore $f$) to be zero.
- Clock the cipher as follows.

$$\begin{aligned} s_0 &:= 0, \\ s_{i+1} &:= suc(nr_i, suc^v(nt_{2i+1}, suc^v(nt_{2i}, s_i))), & i \in \{0, \ldots, 3\} \\ s_{i+5} &:= suc(nr_{i+4}, suc^v(k_{2i+1}, suc^v(k_{2i}, s_{i+4}))), & i \in \{0, \ldots, 3\} \end{aligned}$$

where $v = 1$ for SM and $v = 3$ for CM.

# Initialization

- Let $nt \in (\mathbb{F}_2^8)^8$ be a tag nonce, $nr \in (\mathbb{F}_2^8)^8$ a reader nonce and $k \in (\mathbb{F}_2^8)^8$ be the shared key between the tag and the reader.
- Initialize the registers $l$, $m$, $r$ and $f$ (for SM, ignore $f$) to be zero.
- Clock the cipher as follows.

$$
\begin{aligned}
s_0 &:= 0, \\
s_{i+1} &:= suc(nr_i, suc^v(nt_{2i+1}, suc^v(nt_{2i}, s_i))), && i \in \{0, \dots, 3\} \\
s_{i+5} &:= suc(nr_{i+4}, suc^v(k_{2i+1}, suc^v(k_{2i}, s_{i+4}))), && i \in \{0, \dots, 3\}
\end{aligned}
$$

where $v = 1$ for SM and $v = 3$ for CM.

# Authentication

Let $at \in (\mathbb{F}_2^4)^{16}$ be the tag authenticators and $ar \in (\mathbb{F}_2^4)^{16}$ the reader authenticators.

## SM Authentication

$$
\begin{aligned}
s_i &:= suc^2(0, s_{i-1}), & i \in \{9, \ldots, 40\}. \\
at_i &:= output(s_{2i+9}), & \\
at_{i+1} &:= output(s_{2i+10}), & i \in \{0, 2, \ldots, 14\}, \\
ar_i &:= output(s_{2i+11}), & \\
ar_{i+1} &:= output(s_{2i+12}), & i \in \{0, 2, \ldots, 14\}.
\end{aligned}
$$

# Cont.

## CM Authentication

$$
\begin{aligned}
s_9 &:= suc^5(0, s_8), \qquad s_{10} := suc(0, s_9), \\
s_i &:= suc^6(0, s_{i-1}), \quad i \in \{11, 13, \ldots, 23\}; \\
s_i &:= suc(0, s_{i-1}) \qquad i \in \{12, 14, \ldots, 24\}; \\
s_i &:= suc(0, s_{i-1}) \qquad i \in \{25, 26, \ldots, 38\}; \\
ar_i &:= output(s_{i+9}) \qquad i \in \{0, 1, \ldots, 15\}; \\
at_0 &:= 0xf, \qquad at_1 := 0xf, \\
at_i &:= output(s_{i+23}) \qquad i \in \{2, 3, \ldots, 15\}.
\end{aligned}
$$

# The Attack Setting

- The attacker can only capture some *random known* frames with random nonces, he cannot choose the frames with the nonces satisfying some specific properties, e.g. some special differences.
- The techniques requiring chosen nonces, e.g. the differential-like chosen nonces attacks and the cube attacks and dynamic cube attacks will not work in this realistic setting.
- Fast correlation attacks which usually require large amounts of keystream will not work.

# Previous Results

- For SM, given 1 frame, recover the key in $2^{29.8}$ cipher ticks.
- For CM, given 2640 frames, recover the key in $2^{58}$ cipher ticks.


Flavio D. Garcia, Peter van Rossum, Roel Verdult and Ronny Wichers Schreur.
*Dismantling SecureMemory, CryptoMemory and CryptoRF.*
17th ACM Conference on Computer and Communications Security-CCS'2010, pp. 250-259, 2010, ACM Press. also available at `http://eprint.iacr.org/2010/169`.

# Our Work

Table: Random known nonces key recovery attacks on SecureMemory

| | frames | time | success probability | running time |
|---|---|---|---|---|
| previous attack | 1 | $2^{39.4}$ | 0.57 | minutes |
| **this paper** | 1 | $2^{29.8}$ | 0.75 | seconds |

# Our Work

Table: Random known nonces key recovery attacks on CryptoMemory (success probability 0.5) on 200 CPU cores

| | Theoretical | | | Practical | |
|---|---|---|---|---|---|
| | frames | time | memory | time | memory |
| previous attack | 2640 | $2^{58}$ | $O(2^{32})$ | several weeks | 16 GB |
| **this paper** | 30 | $2^{50}$ | $O(2^{24})$ | several days | 530 MB |

# Our Techniques for SM

- Only make an exhaustive search of the shortest right-most register
- Use the optimal Viterbi-like decoding techniques to recover the internal states of the other registers.
- Exploit the differences in diffusion speeds of the cells of the registers to restore the internal state efficiently.
- Start from the most dense part of the known keystream segment of the left register and fill the gap of 2-step update for adjacent known keystream nibbles.
- Verified by experiments.

# Our Techniques for SM

- Only make an exhaustive search of the shortest right-most register
- Use the optimal Viterbi-like decoding techniques to recover the internal states of the other registers.
- Exploit the differences in diffusion speeds of the cells of the registers to restore the internal state efficiently.
- Start from the most dense part of the known keystream segment of the left register and fill the gap of 2-step update for adjacent known keystream nibbles.
- Verified by experiments.

# Our Techniques for SM

- Only make an exhaustive search of the shortest right-most register
- Use the optimal Viterbi-like decoding techniques to recover the internal states of the other registers.
- Exploit the differences in diffusion speeds of the cells of the registers to restore the internal state efficiently.
- Start from the most dense part of the known keystream segment of the left register and fill the gap of 2-step update for adjacent known keystream nibbles.
- Verified by experiments.

# Our Techniques for SM

- Only make an exhaustive search of the shortest right-most register
- Use the optimal Viterbi-like decoding techniques to recover the internal states of the other registers.
- Exploit the differences in diffusion speeds of the cells of the registers to restore the internal state efficiently.
- Start from the most dense part of the known keystream segment of the left register and fill the gap of 2-step update for adjacent known keystream nibbles.
- Verified by experiments.

# Our Techniques for SM

- Only make an exhaustive search of the shortest right-most register
- Use the optimal Viterbi-like decoding techniques to recover the internal states of the other registers.
- Exploit the differences in diffusion speeds of the cells of the registers to restore the internal state efficiently.
- Start from the most dense part of the known keystream segment of the left register and fill the gap of 2-step update for adjacent known keystream nibbles.
- Verified by experiments.

# Our Techniques for SM

- Only make an exhaustive search of the shortest right-most register
- Use the optimal Viterbi-like decoding techniques to recover the internal states of the other registers.
- Exploit the differences in diffusion speeds of the cells of the registers to restore the internal state efficiently.
- Start from the most dense part of the known keystream segment of the left register and fill the gap of 2-step update for adjacent known keystream nibbles.
- Verified by experiments.

# Our Techniques for CM

- Start from the 16 consecutive keystream nibbles.
- Regard the known intermediate output of the registers as the observed events of the corresponding internal hidden states.
- Partially determine chunks of the state with low complexity by an analysis of the state update function and the output function of the underlying register.
- The positions of the recovered chunks are chosen in such a way that we can determine the maximum keystream information solely based on these states.
- Start from the carefully chosen point in time.
- Verified by experiments.

# Our Techniques for CM

- Start from the 16 consecutive keystream nibbles.
- Regard the known intermediate output of the registers as the observed events of the corresponding internal hidden states.
- Partially determine chunks of the state with low complexity by an analysis of the state update function and the output function of the underlying register.
- The positions of the recovered chunks are chosen in such a way that we can determine the maximum keystream information solely based on these states.
- Start from the carefully chosen point in time.
- Verified by experiments.

# Our Techniques for CM

- Start from the 16 consecutive keystream nibbles.
- Regard the known intermediate output of the registers as the observed events of the corresponding internal hidden states.
- Partially determine chunks of the state with low complexity by an analysis of the state update function and the output function of the underlying register.
- The positions of the recovered chunks are chosen in such a way that we can determine the maximum keystream information solely based on these states.
- Start from the carefully chosen point in time.
- Verified by experiments.

# Our Techniques for CM

- Start from the 16 consecutive keystream nibbles.
- Regard the known intermediate output of the registers as the observed events of the corresponding internal hidden states.
- Partially determine chunks of the state with low complexity by an analysis of the state update function and the output function of the underlying register.
- The positions of the recovered chunks are chosen in such a way that we can determine the maximum keystream information solely based on these states.
- Start from the carefully chosen point in time.
- Verified by experiments.

# Our Techniques for CM

- Start from the 16 consecutive keystream nibbles.
- Regard the known intermediate output of the registers as the observed events of the corresponding internal hidden states.
- Partially determine chunks of the state with low complexity by an analysis of the state update function and the output function of the underlying register.
- The positions of the recovered chunks are chosen in such a way that we can determine the maximum keystream information solely based on these states.
- Start from the carefully chosen point in time.
- Verified by experiments.

# Our Techniques for CM

- Start from the 16 consecutive keystream nibbles.
- Regard the known intermediate output of the registers as the observed events of the corresponding internal hidden states.
- Partially determine chunks of the state with low complexity by an analysis of the state update function and the output function of the underlying register.
- The positions of the recovered chunks are chosen in such a way that we can determine the maximum keystream information solely based on these states.
- Start from the carefully chosen point in time.
- Verified by experiments.

# Our Techniques for CM

- Start from the 16 consecutive keystream nibbles.
- Regard the known intermediate output of the registers as the observed events of the corresponding internal hidden states.
- Partially determine chunks of the state with low complexity by an analysis of the state update function and the output function of the underlying register.
- The positions of the recovered chunks are chosen in such a way that we can determine the maximum keystream information solely based on these states.
- Start from the carefully chosen point in time.
- Verified by experiments.

# Three Phases of The Implementation

1. finding a possible good frame and recovering the left-right pairs;
2. recovering the full internal state $s_8$;
3. recovering the full key from $s_8$.

- Phase 1 is implemented on a single core (of an Intel Core 2 Duo 6600, 2.4 GHz). It takes about 10 minutes to find a possible good frame and recover the possible left-right state pairs subsequently.
- Phase 2 is implemented on a computing cluster with 200 cores (of Intel Xeon L5640, 2.26 GHz). It takes roughly $2-6$ days to find the full internal state (this requires trying several possible good frames found in stage 1).
- Phase 3 is implemented on a single core. It takes on average 2 hours to recover the full secret key from $s_8$.

# One Instance

- Obtain 30 authentication frames from the reference implementation of CM.
- Set $T_r = 54$ for register $r$, $T_l = 45$ and $T_l' = 48$ for register $l$.
- There are 6 possible proper candidates.
- The attack succeeded during the 4th frame.
- Analysis of the 4th frame took about 20.4 hours to find 1 possible candidate state of $s_8$, while analysis of the 3 other frames took several days in total.

# Cont.

- The 4th frame

$$nr = \texttt{0xa8becfc790ce1272}, \quad nt = \texttt{0x8bd5987bdf33aec7},$$
$$ar = \texttt{0x2e0ba95f84eb0a50}, \quad at = \texttt{0xff3f26fab2fb809e},$$

  was found for which there were around $2^{20.73}$ left-right state pairs.
- For each left-right state pair, $2^{27.2}$ inverse cipher ticks are done on average.
- The secret key `0xf7fb3e25ab1c74d8` was found for the state

$$s_8 = (\texttt{0x071d0308081a0e}, \texttt{0x1627033e566b74}, \texttt{0x1e1a100e1b}, \texttt{0x0109})$$

# An Inherent Property of CM

## Property of CM

The number of non-coincidence bits between the two intermediate outputs generated by one possible left-right state pair is a fixed constant, if the sum of the numbers of coincidence bits between each one of the intermediate output and the 64-bit keystream is a constant.

1. checked $10^6$ times in the experiments, it holds all the time.
2. The time complexity cannot be further reduced by setting a larger $T_l$. Since in such cases, the entropy of the middle register also increases.
3. Explains why we set $T_l = 45$ and $T_l' = 48$, for we have to discard the pairs resulting in high entropy middle register.

# Conclusions

1. Even the strongest version of the Atmel cipher succumbs to practical attacks using relatively few captured authentication frames.

2. 

| | key | data | time |
|---|---|---|---|
| KeeLoq | 64 | $2^{16}$ known plaintexts | $2^{44.5}$ |
| CryptoMemory | 64 | 30 known frames | $2^{50}$ |

3. Such proprietary ciphers fail to provide enough security even from a practical point of view.

Thank you!

Q & A