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

Alex Biryukov, Ilya Kizhvatov, and Bin Zhang

University of Luxembourg
Faculty of Science, Technology and Communication
6, rue Richard Coudenhove-Kalergi, L-1359 Luxembourg
alex.biryukov@uni.lu

Abstract. SecureMemory (SM), CryptoMemory (CM) and CryptoRF (CR) are the Atmel chip families with wide applications in practice. They implement a proprietary stream cipher, which we call the Atmel cipher, to provide authenticity, confidentiality and integrity. At CCS'2010, it was shown that given 1 keystream frame, the secret key in SM protected by the simple version of the cipher can be recovered in $2^{39.4}$ cipher ticks and if 2640 keystream frames are available, the secret key in CM guarded by the more complex version of the cipher can be restored in 2^{58} cipher ticks. In this paper, we show much more efficient and practical attacks on both versions of the Atmel cipher. The idea is to dynamically reconstruct the internal state of the underlying register by exploiting the different diffusion speeds of the different cells. For SM, we can recover the secret key in $2^{29.8}$ cipher ticks given 1 keystream frame; for CM, we can recover the secret key in 2^{50} cipher ticks with around 24 frames. Practical implementation of the full attack confirms our results.

Keywords: Stream ciphers, RFID, Frame, SecureMemory, CryptoMemory

1 Introduction

The Atmel Cipher. The Atmel chips AT88SC153 and AT88SC1608, called SecureMemory (SM) family, were introduced in 1999 [3]. The CryptoMemory (CM) family including the AT88SCxxxxC chips was introduced in early 2002 [2] with more advanced cryptographic features. These two families are ISO/IEC 7816 smart cards that communicate through a contact interface; in late 2003, the CryptoRF (CR) family (ISO/IEC 14443-B smart cards) [10], which is a variant of the CM family with the AT88SCxxxxCRF chips was introduced with a RF interface. These chips are widely used in practice all over the world, e.g. in smart cards (ID and access cards, health care cards, loyalty cards, Internet kiosks, energy meters and e-government) [4], in the widely sold NVIDIA graphics

* This is the full version of the paper with the same title published at the 9th International Conference on Applied Cryptography and Network Security-ACNS 2011, Springer-Verlag, LNCS vol. 6715, pp. 91-109, last revised December 23, 2011.

cards [12], the Microsoft's Zune Player [7] and SanDisk's Sansa Connect [13]. For more complete and detailed information, please see [4, 8].

A proprietary stream cipher, which we call the Atmel cipher, lies at the core of the chips to provide authenticity, confidentiality and integrity. The SM family uses the simple version of the cipher, while in CM and CR, the more complex version of the cipher is adopted. The difference between the simple version and the complex one is that there is 1 byte feedback of the output into the other shift registers every cipher tick in the complex version. Besides, in CM, the initialization phase and the generation of the authenticators is much more complicated. There are more mixing rounds in CM before the output is produced. It is commonly believed that the complex version of the Atmel cipher provides much stronger security.

Previous work. At CCS'2010, the Atmel cipher is described and analyzed in the authentication mechanism [8]. In such a scenario, the tag and the reader exchange their 64-bit nonces and use the shared 64-bit key to generate some keystream nibbles as authenticators. The attacker is assumed to be able to capture some keystream frames produced by the same shared key, but with different nonces. Throughout this paper, we call the result of a single authentication session (with 128-bit keystream) a keystream frame, or briefly a frame. In [8], it was shown that there exists a key recovery attack on SM in $2^{39.4}$ cipher ticks with probability 0.57, given 1 frame and a key recovery attack on CM in 2^{58} cipher ticks[1], if 2640 frames are available. Hence, in theory, both versions of the cipher do not provide the full security with respect to their key length. However, in practice, the challenge is how to efficiently break the cipher with as few frames as possible? Are there any cryptanalytic techniques that could be used in such a *restricted* setting? Note that in our scenario, 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. Thus, the techniques requiring chosen nonces, e.g. the differential-like chosen nonces attacks and the cube attacks [5, 6] will not work in this realistic setting, neither will fast correlation attacks [11] which usually require large amounts of keystream.

Our contribution. In this paper, we present practical *random known* nonces attacks on both version of the Atmel cipher. In contrast to the attack in [8], which had to exhaustively search the left and right registers for each captured frame, our attack only makes an exhaustive search of the shortest right-most register and uses the optimal Viterbi-like decoding techniques [14] to recover the internal states of the other registers. We exploit the differences in diffusion speeds of the cells of the registers to restore the internal state efficiently. For SM, by starting from the most dense part of the known keystream segment of the left register, our technique can fill the gap of 2-step update for adjacent known keystream nibbles very well, resulting in a key recovery attack in $2^{29.8}$ cipher ticks with

[1] In [8], this complexity is claimed to be 2^{52} cipher ticks, however, the complexity of unrolling the cipher 64 steps is ignored [9].

success probability 0.75, given 1 frame. This is about 1000 times faster than that in [8]. For CM, by a careful analysis of the state update function and the output function of the underlying register, we can partially determine chunks of the state with low complexity. 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. By starting from the carefully chosen point in time, we mount an attack on CM in 2^{50} cipher ticks with around 24 frames, which is 2^8 times faster than that in [8] and uses much less frames. The extremely low data complexity of our attack makes it more threatening in practice, an attacker can easily get such a number of frames to mount the attack.

We have fully implemented our attack. It takes several minutes to find a good frame among the 30 given frames and recover the possible left-right state pairs subsequently. Then roughly $2-6$ days are needed to restore the full internal state of a good frame just after the initialization using the 200 CPU cores and another 2 hours on a single core are taken for the full key recovery. The short running time allowed us to run the full attack several times for different keys.

Tables 1 and 2 present a comparison of our new attacks and the attacks of [8][2] on SM and CM respectively. We note that due to the properties of our attack the use of bit-slicing techniques, which according to [9] were employed in [8], is not efficient in its implementation. Our original estimates were also very optimistic: the recovery of just the middle register took only 6 hours and complexity of the total attack seemed around $2^{45}-2^{47}$ cipher ticks, but full implementation has shown dependencies between all the phases of the attack, raising the total complexity to about $2^{49}-2^{50}$ cipher ticks.

Table 1. Key recovery attacks on SecureMemory

	data, frames	time	success probability	running time
attack of [8]	1	$2^{39.4}$	0.57	minutes
this paper	1	$2^{29.8}$	0.75	seconds

Table 2. Key recovery attacks on CryptoMemory (success probability 0.5)

	Theoretical			Practical	
	data, frames	time	memory	running time (200 CPU cores)	memory
attack of [8]	2640	2^{58}	$O(2^{32})$	several weeks (estimated)[9]	16 GB
this paper	30	2^{50}	$O(2^{24})$	several days	530 MB

[2] The authors have corrected their original complexity of 2^{52} (for which the attack runs 2 days) to 2^{58} which probably means increase to several months. However, the authors of [9] used bit-slice implementation which offers some speedup. Our implementation currently is not bit-sliced.

Organization of the paper. We describe the two versions of the Atmel cipher in Section 2 together with the concrete authentication protocols. Our attack on SM is provided in Section 3 and the attack on CM is given in Section 4. We describe our practical implementation of the full key recovery for CM in Section 5 and give our conclusions in Section 6.

2 Description of the Atmel Cipher and the Authentication Protocol

In this section, we present a description of the two versions of the Atmel cipher together with the concrete authentication protocols. Let us first specify the notations used hereafter.

- $\mathbb{F}_2^n = \{0, 1, \ldots, 2^n - 1\}$.
- $(x_0 x_1 \ldots x_{n-1}) \in \mathbb{F}_2^n$ with x_0 being the most and x_{n-1} being the least significant bits.
- suc is the state transition function.
- $a \in \mathbb{F}_2^8$ is the input to the state.
- the cipher state s and the successor state $s' = suc^1(a, s) = suc(a, s)$.
- $suc^n(a, s) = suc^{n-1}(a, suc(a, s))$ for $n > 1$.
- the left register $l = (l_0, l_1, \ldots, l_6) \in (\mathbb{F}_2^5)^7$.
- the middle register $m = (m_0, m_1, \ldots, m_6) \in (\mathbb{F}_2^7)^7$.
- the right register $r = (r_0, r_1, \ldots, r_4) \in (\mathbb{F}_2^5)^5$.
- the feedback register $f = (f_0, f_1) \in (\mathbb{F}_2^4)^2$ for CM.
- $L : \mathbb{F}_2^n \to \mathbb{F}_2^n$ is the bitwise left rotation defined by $L(x_0 x_1 \ldots x_{n-1}) = (x_1 \ldots x_{n-1} x_0)$.
- $+$ is the integer addition.
- the modified modular addition $\boxplus : \mathbb{F}_2^n \times \mathbb{F}_2^n \to \mathbb{F}_2^n$ is defined as:

$$x \boxplus y = \begin{cases} x + y \pmod{2^n - 1} & \text{if } x = y = 0 \text{ or } x + y \neq 0 \pmod{2^n - 1} \\ 2^n - 1 & \text{otherwise} \end{cases}$$

2.1 Specification of the Atmel Cipher

Both versions consist of 3 shift registers, i.e., the left register l, the middle register m and the right register r. The complex version in CM has an additional feedback register f to store the last 8 bits of output. The cipher structure is illustrated in Fig. 1. 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. First 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 $b = a$. Then, $\hat{l}_i = l_i$, $\hat{m}_j = m_j$ and $\hat{r}_k = r_k$ for $i \neq 2$, $j \neq 4$ and $k \neq 1$. For $i = 2$, $j = 4$ and $k = 1$,

$$\hat{l}_2 := l_2 \oplus b_3 b_4 b_5 b_6 b_7, \quad \hat{m}_4 := m_4 \oplus b_4 b_5 b_6 b_7 b_0 b_1 b_2, \quad \hat{r}_1 := r_1 \oplus b_0 b_1 b_2 b_3 b_4.$$

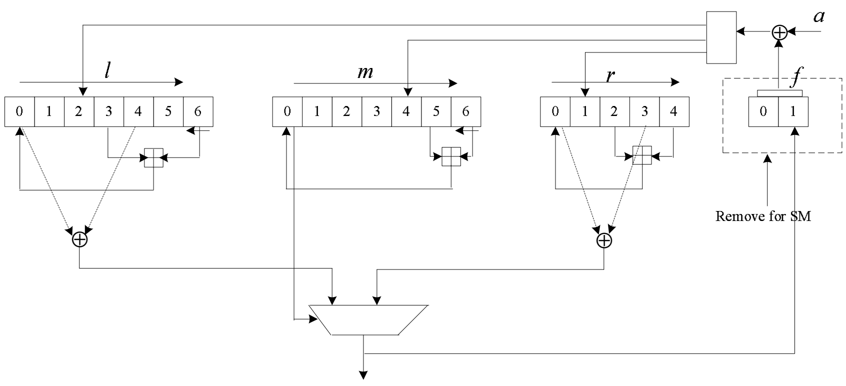

Fig. 1. The ciphers

Second, 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 $\boxplus$.

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

Finally, 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 $\|$ be the concatenation operation, denote by $outputl(l') = l'_{0,1} \oplus l'_{4,1} \parallel l'_{0,2} \oplus l'_{4,2} \parallel l'_{0,3} \oplus l'_{4,3} \parallel l'_{0,4} \oplus l'_{4,4}$ 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

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

Note that the rightmost 4 bits of m'_0 selects either a bit from $outputl(l')$ or a bit from $outputr(r')$ to be output as the keystream bit. For CM, let $f'_0 = \hat{f}_1 = f_1$ and $f'_1 = output(s')$.

2.2 The Authentication Protocol

In the protocol, the tag and the reader exchange the nonces (depicted in Fig. 2) and use the cipher to generate keystream that will be used as authenticators for both sides.

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. First initialize all the registers l, m, r and f (for SM, ignore f) to be zero, then the cipher is clocked as follows.

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

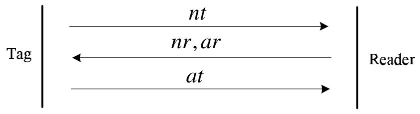

Fig. 2. The authentication protocol

$$s_{i+5} := suc(nr_{i+4}, suc^v(k_{2i+1}, suc^v(k_{2i}, s_{i+4}))), \quad i \in \{0, \ldots, 3\}$$

where $v = 1$ for SM and $v = 3$ for CM. Table 3 shows the schematic view of the input in the initialization phase. Note that the states $s_0, \ldots, s_7, s_8$ are

Table 3. The injection procedures

	SM	CM
s_0	nt_0, nt_1, nr_0	nt_0, nt_0, nt_0 nt_1, nt_1, nt_1, nr_0
s_1	nt_2, nt_3, nr_1	nt_2, nt_2, nt_2 nt_3, nt_3, nt_3, nr_1
s_2	nt_4, nt_5, nr_2	nt_4, nt_4, nt_4 nt_5, nt_5, nt_5, nr_2
s_3	nt_6, nt_7, nr_3	nt_6, nt_6, nt_6 nt_7, nt_7, nt_7, nr_3
s_4	k_0, k_1, nr_4	k_0, k_0, k_0 k_1, k_1, k_1, nr_4
s_5	k_2, k_3, nr_5	k_2, k_2, k_2 k_3, k_3, k_3, nr_5
s_6	k_4, k_5, nr_6	k_4, k_4, k_4 k_5, k_5, k_5, nr_6
s_7	k_6, k_7, nr_7	k_6, k_6, k_6 k_7, k_7, k_7, nr_7
s_8		

non-consecutive. There are 24 setup rounds for SM and 56 setup rounds for CM respectively. Let $at \in (\mathbb{F}_2^4)^{16}$ be the tag authenticators and $ar \in (\mathbb{F}_2^4)^{16}$ the reader authenticators. The precise definitions of the authentication process are given as follows.

SM Authentication Define the following states and outputs:

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

CM Authentication Define the following states and outputs:

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

Note that there are 16 consecutive keystream nibbles in the frame, i.e., ar_{14}, ar_{15}, $at_2, at_3, \ldots, at_{15}$. Since CM and CR use the same version of the cipher, we will ignore this distinction hereafter. The attacker's aim is to restore the shared 64-bit key from a number of captured frames, for each of which only a short keystream segment is known.

3 Our Attack on SecureMemory

First note that from (1), for $i \in \{0, 1, 2, 3\}$ we have

$$output(s')_i = m'_{0,i+3} \cdot outputr(r')_i \oplus (1 \oplus m'_{0,i+3}) \cdot outputl(l')_i. \tag{2}$$

This suggests that $P\big(output(s')_i = outputr(r')_i\big) = \frac{3}{4}$ and $P\big(output(s')_i = outputl(l')_i\big) = \frac{3}{4}$. Thus, we can make an exhaustive search of all the possible s_8 states of the right register r and use a classical correlation test to find the correct state. This is feasible because we can run r independently of l and m. Note that there are 128 bits of known keystream from 1 SM frame. The correct state of r could pass the test with probability $\sum_{i=T_r}^{128} \binom{128}{i} (\frac{3}{4})^i (\frac{1}{4})^{128-i}$ and a wrong guess would pass with probability $\sum_{i=T_r}^{128} \binom{128}{i} (\frac{1}{2})^{128}$, where T_r is the threshold value of the correlation test.

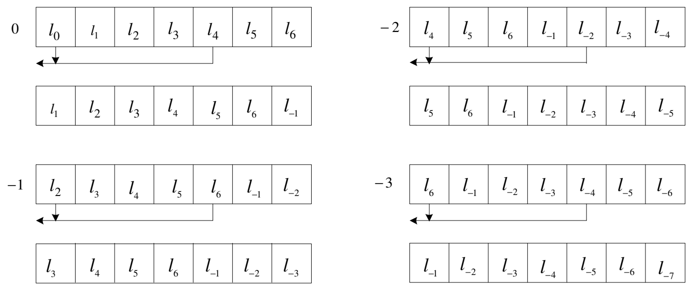

Fig. 3. The backward diffusion of l in SM

Then, unlike the attack in [8], we do not make a second exhaustive search of the left register l. To get the candidate states of l matching the keystream bits where the intermediate output $outputr(r')$ of r does not generate the correct bit (There are 32 such bits on average), we regard the known keystream bits generated by l as the observed events of the internal hidden states, as in the classical Viterbi decoding scenario [14]. However, we found that it is not easy to directly use the Viterbi decoding algorithm here. Instead, we exploit the different diffusion speeds of the different cells in l to enumerate the possible candidates dynamically. Fig. 3 and 4 show the diffusion process of the cells in l. Let l_i $(i \geq 0)$ be the 5-bit content of the corresponding cell, then for every second step i, we know some bits of the xor of the 0th cell and the 4th cell. In fact, the bits are distributed according to the xor between $outputr(r')$ and at and ar. At some instance, it may happen that there are no bits known; while at other step, it also may happen that we know the whole nibble.

Let $lk_i = outputl(l')$ at step i, though sometimes lk_i is unknown. Our observation is that some cells of the initial state affect the output more often than others. Hence, if we isolate the low-effect cells and first determine the cells that have the most extensive effect on the output, it is expected that in this way, we need not try all the possible states one-by-one. Besides, we can shift the starting point of our decoding algorithm. The chosen criterion is determined by the problem that which cells we choose to determine first. We have the following theorem, proved in Appendix A, on the latter problem. For any starting point, define the starting state to be $ls_0 = \{l_0, l_1, l_2, l_3, l_4, l_5, l_6\}$. The counting of the predecessor and successor states and the lk_is follow Fig. 3 and 4.

Theorem 1. *For any starting state ls_0, if we choose $A = \{l_0, l_1, l_3, l_4, l_6\}$ to determine first, then $\{lk_3, lk_5, lk_7, lk_9, lk_{11}\}$ depend on l_8 and A, $\{lk_{-3}, lk_0, lk_2, lk_4\}$ only depend on A, $\{lk_{-1}, lk_6\}$ depend on A and l_2, and $\{lk_1, lk_8, lk_{10}\}$ depend on A, l_2 and l_8.*

Fig. 4. The forward diffusion of l in SM

Theorem 1 shows that if we know A, then we can reduce the possible values of l_8 to a large extent, for there are 5 equations that could be used for check. After determining l_8 and A, other cells of the state ls_0 could be restored easily, for we can just run l forward and backward from ls_0 to get the candidates and then clock back from ls_0 to restore the real initial state. This theorem also indicates how to choose the starting state. Let $N_H(lk_i)$ be number of known bits in lk_i for $0 \leq i \leq 31$, then we define the following function:

$$\Psi(i) = \begin{cases} \sum_{j\in\{0,2,3,4,5,7,9,11\}} N_H(lk_{i+j}) & \text{for } 0 \leq i \leq 2 \\ \sum_{j\in\{-3,0,2,3,4,5,7,9,11\}} N_H(lk_{i+j}) & \text{for } 3 \leq i \leq 20 \\ \sum_{j\in\{-3,0,2,3,4,5,7,9\}} N_H(lk_{i+j}) & \text{for } 21 \leq i \leq 24 \\ \sum_{j\in\{-3,0,2,3,4,5\}} N_H(lk_{i+j}) & \text{for } 25 \leq i \leq 26 \\ \sum_{j\in\{-3,0,2,3,4\}} N_H(lk_{i+j}) & \text{for } i = 27 \\ \sum_{j\in\{-3,0,2,3\}} N_H(lk_{i+j}) & \text{for } i = 28 \\ \sum_{j\in\{-3,0,2\}} N_H(lk_{i+j}) & \text{for } i = 29 \\ \sum_{j\in\{-3,0\}} N_H(lk_{i+j}) & \text{for } 30 \leq i \leq 31 \ . \end{cases}$$

Let $I = \max_{0\leq i\leq 31}\Psi(i)$ and $\Psi(J) = I$, then we can start from the state s_{8+J} in the real SM authentication, which will have a maximum reduction effect on the possible candidates. Table 4 shows the distributions of I and J obtained from experiments. From Table 4, we get $P(I \geq 10, J < 25) \approx 0.94$.

Table 4. The distributions of I and J in SM.

I	$I \geq 10$	$I \geq 11$	$I \geq 12$	$I \geq 13$	$I \geq 14$
	0.95	0.87	0.73	0.53	0.34
J	$J \geq 21$	$J \geq 22$	$J \geq 23$	$J \geq 24$	$J \geq 25$
	(0.0297, 0.0344)	(0.0205, 0.028)	(0.003, 0.014)	(0.003, 0.007)	(0.0006, 0.0018)

Now we are ready to enumerate the possible states of A consistent with $\{lk_0, lk_2, lk_4\}$. We first guess l_0 in ls_0, there are 2^5 possibilities. From lk_0, we can get 2^3 candidates of l_4 on average for each l_0. Then from lk_2, we can get around 2 candidates of l_3 for each l_0. Knowing l_0 and l_3, we could derive l_{10}. From lk_4, we could get 2 candidates on average of l_7 for each pair of (l_0, l_3). For each l_7, we could derive one or two (due to the fact that $\boxplus$ is not injective) candidates of l_6. There are 2^5 candidates of l_1, so with a complexity of $2^5 \cdot 4 = 2^7$, we could get $2^5 \cdot 2^5 \cdot 2^3 \cdot 2 \cdot 2 = 2^{15}$ possible combinations of A. Then using the 5 check equations for l_8, we can determine several candidates of l_8 conditioned on A, sometimes even 1. For each possible combination of A and l_8, we can derive the corresponding l_2 from other lk_is. Given l_8 and l_2, we know l_5. Finally, we run l backwards and forwards from ls_0 to further reduce the possibilities. For each surviving candidate for ls_0, run l backwards to recover the real initial state s_8 in the authentication. This procedure has a complexity of $2^{15} \cdot 2^5 \cdot 2^3 \cdot \frac{64}{3} = 2^{27.4}$ cipher ticks. The number of candidates of s_8 of l is directly determined by the coincidence bits between the intermediate output, $outputr(r')$, of r and the keystream. More coincidence bits, more candidates of s_8. From experiments, we found that with probability around 0.92, there are less than or equal to 100 candidates of s_8 recovered. For each right-left candidate pair, we can run the meet-in-the-middle attack in Section 4.2 of [8] to recover the secret key with a complexity of around $2^{24.5}$ cipher ticks. So the total time complexity of our attack is $2^{27.4} + 2^{24.5} + 2^{25} \cdot \frac{64}{3} = 2^{29.8}$ cipher ticks. And the success probability of our attack is around $\sum_{i=T}^{128} \binom{128}{i} (\frac{3}{4})^i (\frac{1}{4})^{128-i} \cdot 0.92 \cdot 0.94$, which is around 0.75 if we set $T = 91$. Given 1 frame, our attack on SM is about 1000 times faster than that in [8] with a higher success rate. This attack is verified on a single CPU core in C. In experiments, it takes tens of seconds to restore the s_8 state of l and r in the real authentication.

4 Our Attack on CryptoMemory

The starting point of our attack on CM is the 16 consecutive keystream nibbles, i.e., we first use the correlation test to find some candidates of the right-most register r. Because of the existence of the feedback register f in CM, we cannot

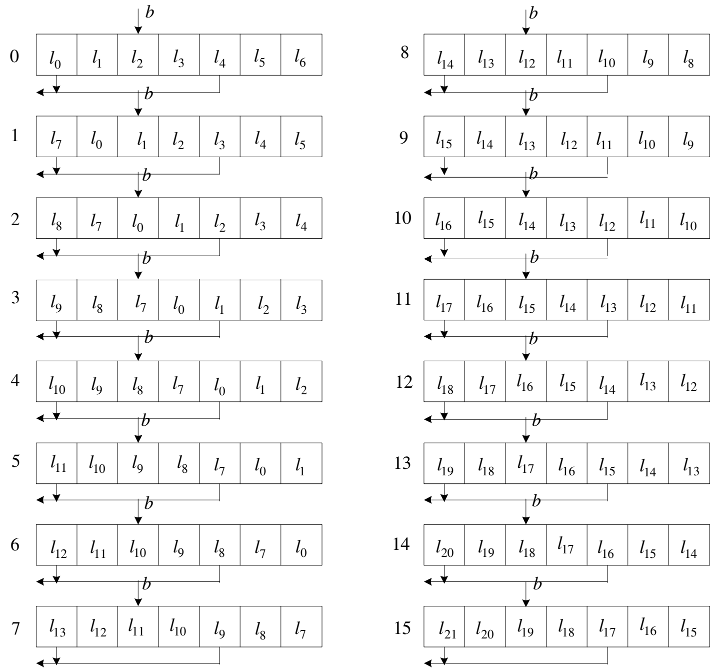

Fig. 5. The forward diffusion of l in CM

run l or r independently in general. But when CM generates the 16 consecutive nibbles, we can run either l or r independently, for in this case we know the feedback bytes, which is the same as the last 64-bit keystream in one frame.

Conditioned on r, we can derive 16-bit information of the intermediate output of l on average by selecting those keystream bits where the intermediate output of r does not generate the correct keystream bit. Again, we regard the known intermediate output of l as the observed events of the corresponding internal hidden states. The good thing for us is that now we know the intermediate output of l for consecutive steps, instead of for every second step. Besides, we need to exploit the properties of the state update function and the output function of l to find the most likely internal states that generate the observed keystream bits. Since both the state update function and the output function of l only depend on very few variables, some of which are shared, we can partially determine chunks of the state with low complexity. Several overlapping partial states could be restored in this way. Then we take an intersection of the overlapping states and use this subset to further reduce the candidates of other parts. The positions of the recovered states are chosen in such a way that we can determine the maximum keystream information solely based on these states. The same techniques are also applied to the middle register m.

4.1 Recovering The Right and Left Registers r and l

Precisely, we first make an exhaustive search of all the possible s_{24} states of the right register r and use a correlation test to filter out the wrong guesses to some extent. The correct candidate could pass the test with probability $\sum_{i=T_r}^{64} \binom{64}{i}(\frac{3}{4})^i(\frac{1}{4})^{64-i}$ and a wrong guess would pass with probability $\sum_{i=T_r}^{64} \binom{64}{i} \cdot (\frac{1}{2})^{64}$, where T_r is the threshold value in correlation test. Since in CM we can only use the 64 consecutive bits for correlation test, instead of 128 bits as in the SM case, we select T_r in such a way that we need not call the following parts of our attack for each captured frame. Our attack is continued only if there are some candidates of r pass the correlation test. If for one frame, there are no output from the correlation test, we discard the frame and try another one.

Now we look at the left register l. We want to restore the s_{24} state of l without exhaustively searching all the possibilities. Still, we did not find a way to use the Viterbi decoding algorithm directly. The following steps have some similarities with the Viterbi algorithm in the sense that we determine the most likely hidden state candidates up to a certain point depending only on the current observable events and the most likely state candidates at the last point. Fig. 5 depicts the forward diffusion process of l. In fact, we can shift the beginning point of our counting, i.e., we can make a chosen time point in the middle to be 0. The chosen criterion is determined by the diffusion properties of the different cells. Again, let lk_i be the intermediate output of l, though sometimes lk_i is unknown. For any starting point, define the starting state to be $ls_0 = \{l_0, l_1, l_2, l_3, l_4, l_5, l_6\}$. We have

$$lk_1 = (l_3 \boxplus L(l_6)) \oplus l_3, \quad lk_2 = (l_2 \boxplus L(l_5)) \oplus l_2, \tag{3}$$
$$lk_3 = (l_1 \boxplus L(l_4)) \oplus l_1, \quad lk_4 = (l_0 \boxplus L(l_3)) \oplus l_0, \tag{4}$$
$$lk_5 = (l_7 \boxplus L(l_2)) \oplus l_7, \quad lk_6 = (l_8 \boxplus L(l_1)) \oplus l_8. \tag{5}$$

The above equations indicate that the output function and the state update function of l depends on very few variables. For example, the 3rd cell of l is used in the current state update function and in the next step output function. Hence, with a complexity of 2^{10}, we can determine the solution set $H_0 = \{l_3, l_6\}$ to (3). Similarly, we derive $H_1 = \{l_2, l_5\}$, $H_2 = \{l_1, l_4\}$, $H_3 = \{l_0, l_3\}$, $H_4 = \{l_7, l_2\}$, $H_5 = \{l_8, l_1\}$ and $H_6 = \{l_9, l_0\}$ from (4) and (5). The cardinality of H_i depends on $N_H(lk_i)$. Note that $H_{0,3} = H_0 \cap H_3 = \{l_3\}$, $H_{1,4} = H_1 \cap H_4 = \{l_2\}$, $H_{2,5} = H_2 \cap H_5 = \{l_1\}$ and $H_{3,6} = H_3 \cap H_6 = \{l_0\}$. Hence, we can reduce the cardinalities of H_i and H_j by keeping only those solutions that have the value patterns existing in the corresponding intersection set $H_{i,j}$. Experiments show that the averaged value of $|H_i|$ is $2^{9.25}$. After reducing the cardinality by intersection, the cardinalities sometimes reduce to half or more, sometimes remain. Then, we combine H_0 with H_3 to get the possible values for $H_{0,3,6} = \{l_0, l_3, l_6\}$. The averaged number of solutions of $H_{0,3,6}$ is $2^{10.96}$ in 2^{20} times of experiments. Note that we have to xor the feedback byte b with the recovered l_0 in H_6 to get the original value of l_0 in ls_0. Similar operation has to be done for the recovered l_1 in H_5 too. To get a maximum reduction effect on the number of possible candidates, we define the following function:

$$\Psi(i) = \sum_{j \in \{1,3,4,8\}} N_H(lk_{i+j}) \quad \text{for } 1 \leq i \leq 7 \ .$$

This function considers the reduction effect on $A = \{l_0, l_1, l_3, l_4, l_6\}$ of the chosen starting state, as can be seen from the diffusion process in Fig. 5. Let $I = \max_{1 \leq i \leq 7} \Psi(i)$ and $\Psi(J) = I$, then we can start from the state s_{24+J} in the real CM authentication to have a maximum reduction effect on A. Since in CM, we know the 16 consecutive lk_is and $\Psi(i)$ is defined over $1 \leq i \leq 7$, we need not build a table similar to Table 4 to list the distributions of I and J, we just use I and J directly following their definitions. From Fig. 5, we have the following theorem, proved in Appendix B.

Theorem 2. *For any starting state ls_0, if we choose $A = \{l_0, l_1, l_3, l_4, l_6\}$ to determine first in CM, then $\{lk_0, lk_1, lk_3, lk_4, lk_7, lk_8, lk_{11}, lk_{15}\}$ depend only on A, $\{lk_{-1}\}$ depends on $\{l_5\}$ and A, and $\{lk_5, lk_{12}\}$ depend on A and l_2.*

Due to the fact that J is not necessarily equal to 1, some of the lk_i could not be used in practice. For example, if $J = 7$, we can only use the lk_is up to $i = 8$. Since we define $\Psi(i)$ only over $1 \leq i \leq 7$, we can use at least $\{lk_0, lk_1, lk_3, lk_4, lk_7, lk_8\}$ for the reduction check of A. Then we use lk_{-1} and lk_5 to reduce the possibilities of H_1. After that, we combine the remaining candidates of H_1 with those of A and run l backwards and forwards to cover the other lk_is. Finally, we clock l back from ls_0 to recover the original initial state, i.e., the state s_{24} in the real CM authentication. Note that the correct candidate of the s_{24} state of l will pass the above procedures with probability 1.

Let n_l be the number of candidates of the s_{24} state of l restored in the above way. Experiments show that n_l is determined by the threshold value T_r in the correlation test of the right-most register r. For example, if we set $T_r = 54$, the averaged value of n_l is approximately $2^{25.4}$, which is very close to $2^{35-64+T_r}$. To further reduce the possible candidates, we resort to the correlation test of l, but restrict ourselves only to the n_l candidates. This step has a complexity of $n^l \cdot \frac{16-2}{3}$, instead of $2^{35} \cdot \frac{16-2}{3} \approx 2^{37.2}$. Let T_l be the threshold value used in the correlation test of the n_l candidates, the correct candidate could pass this test with probability around $P_c = \sum_{i=T_l}^{64} \binom{64}{i} (\frac{3}{4})^i (\frac{1}{4})^{64-i}$, but the wrong candidates would pass with the probability much larger than $\sum_{i=T_l}^{64} \binom{64}{i} (\frac{1}{2})^{64}$. The reason is that the n_l candidates are more likely to be the correct candidate than the ones in the random case, which has 2^{35} possibilities. We use experiments to determine this probability. Let P_w be this false alarm probability, we found that if $T_r = 54$ and $T_l = 45$, then the averaged value of P_w is about 2^{-5}; if $T_r = 54$ and $T_l = 53$, then the averaged value of P_w is about 2^{-19}. We conjecture that in general, P_w is around 2^5 times larger than $\sum_{i=T_l}^{64} \binom{64}{i} (\frac{1}{2})^{64}$. After the correlation test of the n_l candidates, we get about $(n_l - 1) \cdot P_w + 1 \cdot P_c$ candidates for the left register l. Then we turn to the middle register m.

4.2 Recovering The Middle Register m

Now we know the candidate s_{24} (in the real authentication) states of l, r and the real feedback register f, our aim is to restore the candidate s_{24} states of m. By

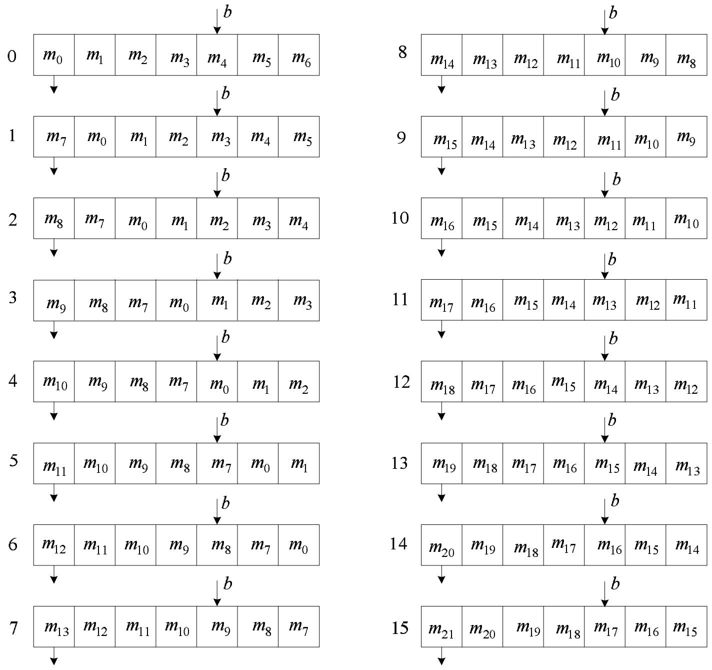

Fig. 6. The forward diffusion process of m in CM

xoring the intermediate outputs $outputl(l')$ of l and those of r, we could know the exact values of the intermediate output of m from (1). Let mk_i denote such value for the step i. Fig. 6 shows the forward diffusion process of m in CM. To make a full use of the known information, we start from the state $s_{24+6} = s_{30}$ in the real authentication. From Fig. 6, we have

$$mk_7 = m_7 \boxplus L(m_0), \tag{6}$$

$$mk_i = m_i \boxplus L(m_{i-1}). \ (8 \le i \le 15) \tag{7}$$

The above equations indicate that the output function and the state update function of the middle register m depend on even fewer variables than l. We can see the content of the feedback cell directly. Hence, with a complexity of 2^{14}, we can determine the candidates of $Q_0 = \{m_7, m_0\}$ satisfying (6). Similarly, we can determine $Q_1 = \{m_8, m_7\}$, $Q_2 = \{m_9, m_8\}$, $Q_3 = \{m_{10}, m_9\}$, $Q_4 = \{m_{11}, m_{10}\}$, $Q_5 = \{m_{12}, m_{11}\}$, $Q_6 = \{m_{13}, m_{12}\}$, $Q_7 = \{m_{14}, m_{13}\}$ and $Q_8 = \{m_{15}, m_{14}\}$ from (7). The cardinality of Q_i depends on $N_H(mk_i)$. Note that $Q_{0,1} = Q_0 \cap Q_1 = \{m_7\}$, $Q_{1,2} = Q_1 \cap Q_2 = \{m_8\}$, $Q_{2,3} = Q_2 \cap Q_3 = \{m_9\}$, $Q_{3,4} = Q_3 \cap Q_4 = \{m_{10}\}$, $Q_{4,5} = Q_4 \cap Q_5 = \{m_{11}\}$, $Q_{5,6} = Q_5 \cap Q_6 = \{m_{12}\}$, $Q_{6,7} = Q_6 \cap Q_7 = \{m_{13}\}$ and $Q_{7,8} = Q_7 \cap Q_8 = \{m_{14}\}$. Experiments show that the averaged value of $|Q_i|$ is $2^{9.1}$ in the random case.

We can reduce the cardinalities of Q_i and Q_{i+1} by keeping only those solutions that have the value patterns existing in the corresponding intersection set $Q_{i,i+1}$. To get a maximum reduction effect, let $I = \min_{0 \le i \le 8} |Q_i|$. Then we start the reduction process from $Q_I = \{m_j, m_k\}$ with $k = 0$ or $k = j - 1$. More precisely, if $I < 8$, we reduce $Q_{I+1} = \{m_{j+1}, m_j\}$ by keeping only those solutions that have the m_j value patterns existing in Q_I. If $I > 0$, we reduce $Q_{I-1} = \{m_k, m_{k-1}\}$ by keeping only those solutions that have the m_k value patterns existing in Q_I. This reduction process is continued to cover Q_0 and Q_8, i.e., we make the reduction step for each Q_i. After reducing each Q_i $(0 \le i \le 8)$, we combine Q_0, Q_1 and Q_2 together to get the possible values for the ith cells $(3 \le i \le 6)$ of state s_{24+6} in the real authentication, i.e., we first fill the 5th and the 6th cells of s_{30} from Q_0, then we fill the 4th cell from the 'm_8' item of Q_1 with the corresponding 'm_7' item equal to the 'm_7' item of Q_0 just filled in. The same procedure applies to the 3rd cell of s_{30}. This is something like a chain. Still, we have to xor the corresponding feedback byte with the recovered m_9 item of Q_2 to get the value of the 3rd cell of state s_{30}.

Then we can check the 3rd to 6th cells of s_{30} immediately by $mk_{20} = m_{14} \boxplus L(m_{13}) = (m_8 \boxplus L(m_7)) \boxplus L(m_7 \boxplus L(m_0))$ and $mk_{21} = m_{15} \boxplus L(m_{14}) = (m_9 \boxplus L(m_8)) \boxplus L(m_8 \boxplus L(m_7))$. Experiments show that with probability around 0.85, the number of the combined 3rd to 6th cells of state s_{30} is less than 2^{18} and in this case the averaged number is $2^{14.6}$, which is much less than 2^{28}. The reduction effect is obvious. Then, we continue the construction of the chain by filling the 0th to 2nd items of the state s_{30} from Q_3, Q_4 and Q_5 respectively. Here we can see the similarity of our method with the Viterbi decoding algorithm. After filling in, we run m from s_{30} backwards and forwards to check all the known mk_is for $0 \le i \le 15$. Finally, we clock back m from s_{30} to recover the s_{24} state in the real CM authentication. Let n_m be the number of candidates of the s_{24} state of m obtained in the above way, our experiments show that with probability around

0.72, n_m is less than 2^{21} and in this case, the averaged number is $2^{18.8}$, which is close to the theoretical value $2^{17} = 2^{49-32}$. Note that in [8], this number is at least $1.43 \cdot 10^9 = 2^{30.41}$. The gain of our method is obvious.

To further reduce the possible candidates, we now run the whole 117-bit state of the cipher backwards to cover the authenticators $ar_{13}, ar_{12}, ar_{11}, ar_{10}, ar_9, ar_8$. Here we have to clock the cipher back 21 steps, i.e., we check those candidates of the middle register that are with the common right-left pair, instead of checking the candidates of the right-left-middle triple. This batch treatment results in a good complexity. In fact, what we do is to regroup all the candidates of the right-left-middle triple of the s_{24} state in the real CM authentication into subgroups which has a common right-left pair and check these subgroups one-by-one. In most cases, we get 1 or 2 candidates of the middle register left corresponding to a right-left state pair, for the above reduction factor is 2^{-24}. If there are no surviving candidates of the middle register we conclude that the corresponding right-left pair is a wrong pair. After this step, there are around $n_m \cdot 2^{-24}$ candidates of m left corresponding to a common right-left state pair.

Next, for each survived candidate for m with the common right-left pair we run the cipher backwards to the s_8 state in the real CM authentication. In this process, we can use the authenticators ar_i for $0 \leq i \leq 7$ to further reduce the possible candidates of m and the right-left pairs. In general, only the correct right-left pair with the correct candidate for the middle register could pass this test, since the reduction factor is 2^{-32}.

4.3 Complexity Analysis

In summary, our attack works in three phases. First, we exhaustively search the shortest register r. In our attack, we set the threshold value T_r of the correlation test in such a way that for some frames, there are no output of the test. Thus, we need not run the following phases of our attack for each frame. Second, we use our method to get the candidates of l without trying all the possible values and for each candidate, we again call the correlation test with a threshold value T_l to further reduce the number of candidates. In this way, it is possible that the correct candidate of l is filtered out. We have to compensate this with more frames.

Definition 1. *We say a frame is a good frame, if both the correct candidate of l and that of r pass the correlation tests in our attack.*

Let $P_l = \sum_{i=T_l}^{T_l'} \binom{64}{i}(\frac{3}{4})^i(\frac{1}{4})^{64-i}$ and $P_r = \sum_{i=T_r}^{64} \binom{64}{i}(\frac{3}{4})^i(\frac{1}{4})^{64-i}$. In theory, with

$$F = \frac{1}{P_l \cdot P_r} = \frac{1}{\sum_{i=T_l}^{T_l'} \binom{64}{i}(\frac{3}{4})^i(\frac{1}{4})^{64-i} \cdot \sum_{i=T_r}^{64} \binom{64}{i}(\frac{3}{4})^i(\frac{1}{4})^{64-i}} \tag{8}$$

frames, we could encounter a good frame to mount our attack. In practice, we usually need more than F frames to get a good frame with probability around 0.5. This is mainly caused by the fact that the left and the right registers are not fully independent from each other. Third, we recover the middle register conditioned on the candidates of l and r. Since there is an unrolling check in this phase, we can determine either the correct right-left-middle state triple or there is no candidate survived, indicating that the frame is not good.

The time complexity of the first phase is $2^{25} \cdot \frac{16-2}{3} = 2^{27.8}$ cipher ticks and we have to repeat this step from each frame until we meet a good frame. Hence, in total, the time complexity in the first phase is $2^{27.8} \cdot F$ cipher ticks. For about $w = \frac{F}{2} \cdot (P_r + (2^{25} - 1) \cdot \sum_{i=T_r}^{64} \binom{64}{i}(\frac{1}{2})^{64})$ frames, we have about $n_r = 1 \cdot P_r + (2^{25} - 1) \cdot \sum_{i=T_r}^{64} \binom{64}{i}(\frac{1}{2})^{64}$ candidates passed from the first phase. In such cases, we invoke the second phase of our attack to find some candidates for the left register. The time complexity of a single run of the second phase of our attack is $C_l = n_l \cdot \frac{16-2}{3} + \frac{2^{10.96} \cdot 2^{10}}{2^{\Psi(J)}} \cdot \frac{2^{10}}{2^{N_H(lk_{-1}) + N_H(lk_5)}}$. We will get about $(n_l - 1) \cdot P_w + 1 \cdot P_c$ candidates after the second phase which is mainly determined by T_l. For each possible combination of the right-left state candidates pair, we check whether the underlying frame is a good one or not in the third phase. The time complexity of a single run of the third phase is at most $C_m = 2^{14} \cdot 9 + 2^{18} \cdot 4 \cdot \frac{16-2}{3} + n_m \cdot 7 + 2 \cdot (64-21)$ cipher ticks. Therefore, the time complexity of our attack so far is around

$$C_{total} = 2^{27.8} \cdot F + w \cdot (n_r \cdot C_l + n_r \cdot ((n_l - 1) \cdot P_w + 1 \cdot P_c) \cdot C_m) \qquad (9)$$

cipher ticks. If we set $T_r = 54$, $T_l = 45$ and $T_l' = 48$, we have $F = 24$ and $C_{total} \approx 2^{50}$ cipher ticks

The success probability of our attack depends on the number of captured frames. Table 5 shows the relation got from 10^6 experiments with randomly generated frames.

Table 5. The success probability of our attack if $T_r \geq 54$ and $T_l = 45$, $T_l' = 48$.

F	24	30	40	45	60	90	120
P_{succ}	0.431	0.505	0.608	0.653	0.759	0.877	0.943

5 Practical Implementation

Here we describe the practical implementation of the full key recovery attack on the CryptoMemory. We have fully implemented our attack. The implementation consists of three stages:

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

The first stage 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. The second stage is the most time-consuming and 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). The last stage is implemented on a single core. It takes on average 2 hours to recover the full secret key from s_8. Note that stage 2 can return several candidates for s_8; in this case we launch stage 3 in parallel on several cores. We will describe our low-memory (compared to that of [8]) key recovery algorithm in more details in Section 5.1.

The low complexity of our attack allowed us to run it several times with different keys. The speed of our programm is about 2^7 clock cycles per inverse cipher tick. Here we present the flow of the attack for one of the runs. We obtained 30 authentication frames from the reference implementation of the Atmel cipher (which was verified against the hardware according to [8]) with the secret key `0xf7fb3e25ab1c74d8`. After this, we proceeded as if we had not known the key. We set $T_r = 54$, $T_l = 45$ and $T_l' = 48$. Then $P_r \approx 0.05$ and $P_l \approx 0.84 - 0.45 = 0.39$. Among the 30 frames we found one possible good frame

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

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 to reduce the number of possible candidates. We launched the second stage of our attack on 200 CPU cores. 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. For this s_8 state, we use our key-recovery technique to restore the key. The secret key `0xf7fb3e25ab1c74d8` was found for the state

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

(each two hexadecimal digits in this notation represent a single register cell).

We note that due the properties of our attack (namely, frequent checking of the cipher output against the keystream while clocking back) its implementation cannot be significantly sped-up by employing a bit-sliced implementation of the cipher, as it was the case in [8] according to [9]. During the experiments, we found an inherent property of CM, i.e., 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. It is checked 10^6 times, the experiments show it holds all the time. This property indicates that we cannot further reduce the time complexity of our attack by setting a larger T_l. Since in such cases, the entropy of the middle register also increases. This property also explains why we set $T_l = 45$ and $T_l' = 48$, for we have to discard the pairs resulting in high entropy middle register.

5.1 Full Key Recovery Using Meet-in-the-Middle

After correctly restoring the s_8 state in the real CM authentication, we need to restore the secret key. The original key recovery stage in [8] for the CM needs 16 GB of storage space[3] and therefore requires a machine with a plenty of RAM to work fast. Motivated by the challenge to make the attack really practical, we present the following low-memory algorithm for the key recovery that could run on a standard machine.

We recall that the original key recovery employs the meet-in-the-middle technique. The first 32 bits of the key are guessed to clock the cipher forward from the s_4 sate in initialization to the state s_6. Then the last 32 bits of the key are guessed to clock the cipher back from s_8 to s_6. The only intersection in s_6 then leads to the correct full key guess. The forward step of this attack results in a table of 2^{32} entries containing the 32-bit guess for the first half of the key and the corresponding value of the 117-bit state s_6.

In our attack, we make use of an observation that allows us to reduce the size of the forward table. Note that after clocking the cipher 3 steps forward, several bytes of the state remain independent of the inputs at these steps. Namely, let us consider the state during initialization just before the injection of k_3, which we will denote by t, and the state right after the injection of k_3 for 3 times, denoted by t'. The following 5 cells of t' do not depend on k_3 and can be expressed in terms of the cells of t.

$$\begin{aligned} l'_2 &= l_3 \boxplus L(l_6) \\ l'_6 &= l_3 \\ m'_2 &= m_5 \boxplus L(m_6) \\ m'_3 &= m_0 \\ m'_4 &= m_1 \end{aligned}$$

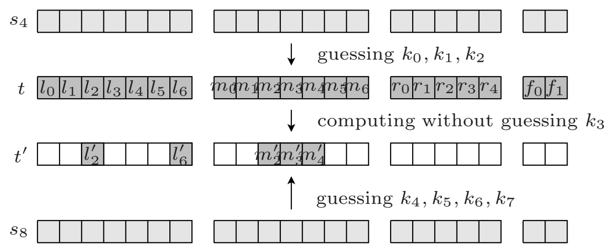

Fig. 7. Low memory meet-in-the-middle for key recovery

Note that these cells comprise 31 bit in total. Now we can shift the meet-in-the-middle point to the state t', looking for matches in the above 5 cells, as

[3] Note that it would be 76 GB in a straightforward way, but according to [9] several techniques were applied to implement key recovery in 16 GB.

shown in Figure 7. The full meet-in-the-middle phase consists of the following steps.

1. In the forward step, we guess the first 24 bits of k and build a table with the entries storing the guess, the corresponding value of the known 5 cells of t' (to perform matching in the next step) and the full state t (to recover k_3 in the last step).
2. In the backward step, we guess the last 32 bits of k, clock the cipher back from s_8 to t' and look for matches in the forward table. For each match, we perform the last step.
3. We guess k_3 and clock the cipher forward from t (taken from the forward table) to t'. If the resulting value matches the one from the backward step, we have found the full key.

Note that our technique is not equivalent to just shifting the meet-in-the-middle point to t and guessing for 5 key bytes in the backward step. Our technique exploits the additional filtering in t' to perform clocking from t to t' only for a portion of guesses and thus has lower complexity.

The memory required for our attack is determined by the size of the forward table, which is $2^{24} \times (3+4+15) = 352$ MB. The time complexity is determined by the backward step and the search for k_3. The backward step requires $2^{32} \times 15$ reverse cipher ticks, and considering states with multiple predecessors (in practice we observed 1.9 times more states after 15 clock-back steps) this is about 2^{37} inverse cipher ticks. The search in the forward table is logarithmic in its size. There will be about 2^{26} matches in t', for each of them $2^8 \cdot 3$ forward cipher ticks are required for the k_3 check, so 2^{36} forward cipher ticks is the added complexity of the k_3 recovery. The total complexity is thus about 2^{37} inverse cipher ticks in the worst case.

Our practical implementation of this key recovery algorithm requires 530 MB of RAM (larger than the ideal figure due to non-packed but handy data structures used) and works on average for 2 hours on a single core of an Intel Core 2 Duo 6600 running at 2.4 GHz. It can be sped up by parallelizing the backward step.

6 Conclusions

In this paper, we have shown practical key recovery attacks on both versions of the Atmel cipher. By using the optimal Viterbi-like decoding techniques to recover the internal states of the left and middle registers and exploiting the different diffusion speeds of the different cells of the underlying registers, our attacks significantly improved the best previously known results [8]. Our analysis shows that even the strongest version of the Atmel cipher succumbs to practical attacks using relatively few captured authentication frames. Our practical implementation recovers the full 64-bit secret key from 30 captured authentication frames in about $2-6$ days using 200 CPU cores. Table 6 shows the comparison of the attack on the Atmel CryptoMemory cipher presented in this paper and that on another proprietary cipher KeeLoq in [1]. One can again conclude that such proprietary ciphers fail to provide enough security even from a practical point of view.

Table 6. Comparison of our attack on the Atmel cipher in CryptoMemory and the attack [1] on KeeLoq

	key length, bits	data complexity	time complexity
KeeLoq	64	2^{16} known plaintexts	$2^{44.5}$
CryptoMemory	64	30 known frames	2^{50}

Acknowledgements. We would like to thank Flavio D. Garcia of Radboud University Nijmegen, the Netherlands for useful discussions.

References

1. W. Aerts, E. Biham, D. de Moitie, E. de Mulder, O. Dunkelman, S. Indesteege, N. Keller, B. Preneel, G. Vandenbosch, and I. Verbauwhede, A practical attack on KeeLoq, Journal of Cryptology, to appear.
2. Atmel. CryptoMemory specification, 2007. 5211A-SMIC-04/07.
3. Jean Pierre Benhammou, Vincent C. Colnot, and David J. Moore. Secure memory device for smart cards, July 2008. US Patent 7395435 B2.
4. Jean Pierre Benhammou and Mary Jarboe. Security at an affordable price. Atmel Applications Journal, 3:29–30, 2004.
5. Itai Dinur, Adi Shamir. Cube Attacks on Tweakable Black Box Polynomials. Advances in Cryptology-EUROCRYPT'2009. LNCS vol. 5479, pp. 278-299. Springer-Verlag 2009.
6. Itai Dinur, Adi Shamir. Breaking Grain-128 with Dynamic Cube Attacks. Fast Software Encryption-FSE'2011. Springer-Verlag, to appear.
7. Brian Dipert. The Zune HD: more than an iPod touch wanna-be? EDN, pp. 20, October 2009.
8. 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://  b  xugw9 b2emmv . ollibeefood.rest/2010/169`.
9. Flavio D. Garcia. Private communication.
10. Mary Jarboe. Introduction to CryptoMemory. Atmel Applications Journal, 3:28, 2004.
11. W. Meier, O. Staffelbach, Fast correlation attacks on certain stream ciphers, Journal of Cryptology, Springer-Verlag,(1989), pp. 159-176.
12. `http://6dp0mbh xh6x6 ek1zvdvd . ollibeefood.rest/downloads/pvzone/Checklist_for_Building_a_HDPC.pdf`
13. `http://d ngm adytdxc nr 2 f6wr. ollibeefood.rest/wiki/SansaConnect`
14. Viterbi AJ. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. IEEE Transactions on Information Theory 13 (2): 260-269, April 1967.
15. Amazon Elastic Compute Cloud (Amazon EC2). `http:// wnm2 9u xza a . ollibeefood.rest/ec2/#pricing`, accessed 22 January 2010.

A Proof of Theorem 1

Proof. It is sufficient to expand the algebraic expressions of k_i by the state update constraints. From Fig. 3 and 4, we have

$$
\begin{aligned}
lk_{-3} &= l_6 \oplus l_{-4},\\
lk_{-2} &= l_4 \oplus l_{-2},\\
lk_{-1} &= l_2 \oplus l_6,\\
lk_0 &= l_0 \oplus l_4,\\
lk_1 &= l_8 \oplus l_2 = (l_2 \oplus L(l_5)) \oplus l_2,\\
lk_2 &= l_{10} \oplus l_0 = (l_0 \oplus L(l_3)) \oplus l_0,\\
lk_3 &= l_{12} \oplus l_8 = (l_8 \oplus L(l_1)) \oplus l_8,\\
lk_4 &= l_{14} \oplus l_{10} = (l_{10} \oplus L(l_7)) \oplus l_{10} = (l_0 \oplus L(l_3) \oplus L(l_3 \oplus L(l_6))) \oplus (l_0 \oplus L(l_3)),\\
lk_5 &= l_{16} \oplus l_{12} = (l_{12} \oplus L(l_9)) \oplus l_{12} = (l_8 \oplus L(l_1) \oplus L(l_1 \oplus L(l_4))) \oplus (l_8 \oplus L(l_1)),\\
lk_6 &= l_{18} \oplus l_{14} = (l_{14} \oplus L(l_{11})) \oplus l_{14} = (l_{10} \oplus L(l_7) \oplus L(l_7 \oplus L(l_2))) \oplus (l_{10} \oplus L(l_7))\\
&= (l_0 \oplus L(l_3) \oplus L(l_3 \oplus L(l_6)) \oplus L(l_3 \oplus L(l_6) \oplus L(l_2))) \oplus (l_0 \oplus L(l_3) \oplus L(l_3 \oplus L(l_6))),\\
lk_7 &= l_{20} \oplus l_{16} = (l_{16} \oplus L(l_{13})) \oplus l_{16} = (l_{12} \oplus L(l_9) \oplus L(l_9 \oplus L(l_0))) \oplus (l_{12} \oplus L(l_9))\\
&= (l_8 \oplus L(l_1) \oplus L(l_1 \oplus L(l_4)) \oplus L(l_1 \oplus L(l_4) \oplus L(l_0))) \oplus (l_8 \oplus L(l_1) \oplus L(l_1 \oplus L(l_4))),\\
lk_8 &= l_{22} \oplus l_{18} = (l_{18} \oplus L(l_{15})) \oplus l_{18} = (l_{14} \oplus L(l_{11}) \oplus L(l_{11} \oplus L(l_8))) \oplus (l_{14} \oplus L(l_{11}))\\
&= (l_{10} \oplus L(l_7) \oplus L(l_7 \oplus L(l_2)) \oplus L(l_7 \oplus L(l_2) \oplus L(l_8))) \oplus (l_{10} \oplus L(l_7) \oplus L(l_7 \oplus L(l_2)))\\
&= (l_0 \oplus L(l_3) \oplus L(l_3 \oplus L(l_6)) \oplus L(l_3 \oplus L(l_6) \oplus L(l_2)) \oplus L(l_3 \oplus L(l_6) \oplus L(l_2) \oplus L(l_8)))\\
&\quad \oplus (l_0 \oplus L(l_3) \oplus L(l_3 \oplus L(l_6)) \oplus L(l_3 \oplus L(l_6) \oplus L(l_2))),\\
lk_9 &= l_{24} \oplus l_{20} = (l_{20} \oplus L(l_{17})) \oplus l_{20} = (l_{16} \oplus L(l_{13}) \oplus L(l_{13} \oplus L(l_{10}))) \oplus (l_{16} \oplus L(l_{13}))\\
&= (l_{12} \oplus L(l_9) \oplus L(l_9 \oplus L(l_0)) \oplus L(l_9 \oplus L(l_0) \oplus L(l_{10}))) \oplus (l_{12} \oplus L(l_9) \oplus L(l_9 \oplus L(l_0)))\\
&= (l_8 \oplus L(l_1) \oplus L(l_1 \oplus L(l_4)) \oplus L(l_1 \oplus L(l_4) \oplus L(l_0)) \oplus L(l_1 \oplus L(l_4) \oplus L(l_0) \oplus L(l_0\\
&\quad \oplus L(l_3)))) \oplus (l_8 \oplus L(l_1) \oplus L(l_1 \oplus L(l_4)) \oplus L(l_1 \oplus L(l_4) \oplus L(l_0))),\\
lk_{10} &= l_{26} \oplus l_{22} = (l_{22} \oplus L(l_{19})) \oplus l_{22} = (l_{18} \oplus L(l_{15}) \oplus L(l_{15} \oplus L(l_{12}))) \oplus (l_{18} \oplus L(l_{15}))\\
&= (l_{14} \oplus L(l_{11}) \oplus L(l_{11} \oplus L(l_8)) \oplus L(l_{11} \oplus L(l_8) \oplus L(l_8 \oplus L(l_1)))) \oplus (l_{14} \oplus L(l_{11})\\
&\quad \oplus L(l_{11} \oplus L(l_8)))\\
&= (l_{10} \oplus L(l_7) \oplus L(l_7 \oplus L(l_2)) \oplus L(l_7 \oplus L(l_2) \oplus L(l_8)) \oplus L(l_7 \oplus L(l_2) \oplus L(l_8)\\
&\quad \oplus L(l_8 \oplus L(l_1)))) \oplus (l_{10} \oplus L(l_7) \oplus L(l_7 \oplus L(l_2)) \oplus L(l_7 \oplus L(l_2) \oplus L(l_8)))\\
&= (l_0 \oplus L(l_3) \oplus L(l_3 \oplus L(l_6)) \oplus L(l_3 \oplus L(l_6) \oplus L(l_2)) \oplus L(l_3 \oplus L(l_6) \oplus L(l_2) \oplus L(l_8))\\
&\quad \oplus L(l_3 \oplus L(l_6) \oplus L(l_2) \oplus L(l_8) \oplus L(l_8 \oplus L(l_1)))) \oplus (l_0 \oplus L(l_3) \oplus L(l_3 \oplus L(l_6))\\
&\quad \oplus L(l_3 \oplus L(l_6) \oplus L(l_2)) \oplus L(l_3 \oplus L(l_6) \oplus L(l_2) \oplus L(l_8))),\\
lk_{11} &= l_{28} \oplus l_{24} = (l_{24} \oplus L(l_{21})) \oplus l_{24} = (l_{20} \oplus L(l_{17}) \oplus L(l_{17} \oplus L(l_{14}))) \oplus (l_{20} \oplus L(l_{17}))\\
&= (l_{16} \oplus L(l_{13}) \oplus L(l_{13} \oplus L(l_{10})) \oplus L(l_{13} \oplus L(l_{10}) \oplus L(l_{10} \oplus L(l_7)))) \oplus (l_{16} \oplus L(l_{13})\\
&\quad \oplus L(l_{13} \oplus L(l_{10})))\\
&= (l_{12} \oplus L(l_9) \oplus L(l_9 \oplus L(l_0)) \oplus L(l_9 \oplus L(l_0) \oplus L(l_0 \oplus L(l_3))) \oplus L(l_9 \oplus L(l_0)\\
&\quad \oplus L(l_0 \oplus L(l_3)) \oplus L(l_0 \oplus L(l_3) \oplus L(l_3 \oplus L(l_6))))) \oplus (l_{12} \oplus L(l_9) \oplus L(l_9 \oplus L(l_0))\\
&\quad \oplus L(l_9 \oplus L(l_0) \oplus L(l_0 \oplus L(l_3))))\\
&= (l_8 \oplus L(l_1) \oplus L(l_1 \oplus L(l_4)) \oplus L(l_1 \oplus L(l_4) \oplus L(l_0)) \oplus L(l_1 \oplus L(l_4) \oplus L(l_0) \oplus L(l_0\\
&\quad \oplus L(l_3))) \oplus L(l_1 \oplus L(l_4) \oplus L(l_0) \oplus L(l_0 \oplus L(l_3)) \oplus L(l_0 \oplus L(l_3) \oplus L(l_3 \oplus L(l_6)))))\\
&\quad \oplus (l_8 \oplus L(l_1) \oplus L(l_1 \oplus L(l_4)) \oplus L(l_1 \oplus L(l_4) \oplus L(l_0)) \oplus L(l_1 \oplus L(l_4) \oplus L(l_0)\\
&\quad \oplus L(l_0 \oplus L(l_3)))).
\end{aligned}
$$

□

B Proof of Theorem 2

Proof. It is sufficient to expand the algebraic expressions of k_i by the state update constraints.

$$
\begin{aligned}
lk_{-1} &= l_1 \oplus l_5,\\
lk_0 &= l_0 \oplus l_4,\\
lk_1 &= l_7 \oplus l_3 = (l_3 \oplus L(l_6)) \oplus l_3,\\
lk_2 &= l_8 \oplus l_2 = (l_2 \oplus L(l_5)) \oplus l_2,\\
lk_3 &= l_9 \oplus l_1 = (l_1 \oplus L(l_4)) \oplus l_1,\\
lk_4 &= l_{10} \oplus l_0 = (l_0 \oplus L(l_3)) \oplus l_0,\\
lk_5 &= l_{11} \oplus l_7 = (l_7 \oplus L(l_2)) \oplus l_7 = (l_3 \oplus L(l_6) \oplus L(l_2) \oplus (l_3 \oplus L(l_6)),\\
lk_6 &= l_{12} \oplus l_8 = (l_8 \oplus L(l_1)) \oplus l_8 = (l_2 \oplus L(l_5) \oplus L(l_1) \oplus (l_2 \oplus L(l_5)),\\
lk_7 &= l_{13} \oplus l_9 = (l_9 \oplus L(l_0)) \oplus l_9 = (l_1 \oplus L(l_4) \oplus L(l_0) \oplus (l_1 \oplus L(l_4)),\\
lk_8 &= l_{14} \oplus l_{10} = (l_{10} \oplus L(l_7)) \oplus l_{10} = (l_0 \oplus L(l_3) \oplus L(l_3 \oplus L(l_6))) \oplus (l_0 \oplus L(l_3)),\\
lk_9 &= l_{15} \oplus l_{11} = (l_{11} \oplus L(l_8)) \oplus l_{11} = (l_7 \oplus L(l_2) \oplus L(l_2 \oplus L(l_5))) \oplus (l_7 \oplus L(l_2))\\
&= (l_3 \oplus L(l_6) \oplus L(l_2) \oplus L(l_2 \oplus L(l_5))) \oplus (l_3 \oplus L(l_6) \oplus L(l_2)),\\
lk_{10} &= l_{16} \oplus l_{12} = (l_{12} \oplus L(l_9)) \oplus l_{12} = (l_8 \oplus L(l_1) \oplus L(l_1 \oplus L(l_4))) \oplus (l_8 \oplus L(l_1))\\
&= (l_2 \oplus L(l_5) \oplus L(l_1) \oplus L(l_1 \oplus L(l_4))) \oplus (l_2 \oplus L(l_5) \oplus L(l_1)),\\
lk_{11} &= l_{17} \oplus l_{13} = (l_{13} \oplus L(l_{10})) \oplus l_{13} = (l_9 \oplus L(l_0) \oplus L(l_0 \oplus L(l_3))) \oplus (l_9 \oplus L(l_0))\\
&= (l_1 \oplus L(l_4) \oplus L(l_0) \oplus L(l_0 \oplus L(l_3))) \oplus (l_1 \oplus L(l_4) \oplus L(l_0)),\\
lk_{12} &= l_{18} \oplus l_{14} = (l_{14} \oplus L(l_{11})) \oplus l_{14} = (l_{10} \oplus L(l_7) \oplus L(l_7 \oplus L(l_2))) \oplus (l_{10} \oplus L(l_7))\\
&= (l_0 \oplus L(l_3) \oplus L(l_3 \oplus L(l_6)) \oplus L(l_3 \oplus L(l_6) \oplus L(l_2))) \oplus (l_0 \oplus L(l_3)\\
&\quad \oplus L(l_3 \oplus L(l_6))),\\
lk_{13} &= l_{19} \oplus l_{15} = (l_{15} \oplus L(l_{12})) \oplus l_{15} = (l_{11} \oplus L(l_8) \oplus L(l_8 \oplus L(l_1))) \oplus (l_{11} \oplus L(l_8))\\
&= (l_7 \oplus L(l_2) \oplus L(l_2 \oplus L(l_5)) \oplus L(l_2 \oplus L(l_5) \oplus L(l_1))) \oplus (l_7 \oplus L(l_2)\\
&\quad \oplus L(l_2 \oplus L(l_5)))\\
&= l_3 \oplus L(l_6) \oplus L(l_2) \oplus L(l_2 \oplus L(l_5)) \oplus L(l_2 \oplus L(l_5) \oplus L(l_1))) \oplus (l_3 \oplus L(l_6) \oplus L(l_2)\\
&\quad \oplus L(l_2 \oplus L(l_5))),\\
lk_{14} &= l_{20} \oplus l_{16} = (l_{16} \oplus L(l_{13})) \oplus l_{16} = (l_{12} \oplus L(l_9) \oplus L(l_9 \oplus L(l_0))) \oplus (l_{12} \oplus L(l_9))\\
&= (l_8 \oplus L(l_1) \oplus L(l_1 \oplus L(l_4)) \oplus L(l_1 \oplus L(l_4) \oplus L(l_0))) \oplus (l_8 \oplus L(l_1)\\
&\quad \oplus L(l_1 \oplus L(l_4)))\\
&= (l_2 \oplus L(l_5) \oplus L(l_1) \oplus L(l_1 \oplus L(l_4)) \oplus L(l_1 \oplus L(l_4) \oplus L(l_0))) \oplus (l_2 \oplus L(l_5) \oplus L(l_1)\\
&\quad \oplus L(l_1 \oplus L(l_4))),\\
lk_{15} &= l_{21} \oplus l_{17} = (l_{17} \oplus L(l_{14})) \oplus l_{17} = (l_{13} \oplus L(l_{10}) \oplus L(l_{10} \oplus L(l_7))) \oplus (l_{13} \oplus L(l_{10}))\\
&= (l_9 \oplus L(l_0) \oplus L(l_0 \oplus L(l_3)) \oplus L(l_0 \oplus L(l_3) \oplus L(l_3 \oplus L(l_6)))) \oplus (l_9 \oplus L(l_0)\\
&\quad \oplus L(l_0 \oplus L(3)))\\
&= (l_1 \oplus L(l_4) \oplus L(l_0) \oplus L(l_0 \oplus L(l_3)) \oplus L(l_0 \oplus L(l_3) \oplus L(l_3 \oplus L(l_6)))) \oplus (l_1 \oplus L(l_4)\\
&\quad \oplus L(l_0) \oplus L(l_0 \oplus L(3))).
\end{aligned}
$$

□