Cryptanalysis of the DECT Standard Cipher

Karsten Nohl[1], Erik Tews[2], and Ralf-Philipp Weinmann[3]

[1] University of Virginia
nohl@cs.virginia.edu
[2] Technische Universität Darmstadt
e_tews@cdc.informatik.tu-darmstadt.de
[3] University of Luxembourg
ralf-philipp.weinmann@uni.lu

Abstract. The DECT Standard Cipher (DSC) is a proprietary 64-bit stream cipher based on irregularly clocked LFSRs and a non-linear output combiner. The cipher is meant to provide confidentiality for cordless telephony. This paper illustrates how the DSC was reverse-engineered from a hardware implementation using custom firmware and information on the structure of the cipher gathered from a patent. Beyond disclosing the DSC, the paper proposes a practical attack against DSC that recovers the secret key from 2^{15} keystreams on a standard PC with a success rate of 50% within hours; somewhat faster when a CUDA graphics adapter is available.

Keywords: DECT, DECT Standard Cipher, stream cipher, cryptanalysis, linear feedback shift register.

1 Introduction

Cordless phones using the Digital Enhanced Cordless Telecommunications standard (DECT) are among the most widely deployed security technologies with 90 million new handsets shipping every year [1]. However, DECT does not provide sufficient security for its intended application 'cordless telephony' as it fails to deliver confidentiality and access control.

The technology is also popular in other applications with even higher security needs including machine automation, building access control, alarm systems, and wireless credit card terminals [2].

DECT's need for security is covered by two proprietary algorithms: The DECT Standard Authentication Algorithm (DSAA) for authentication [3] and the DECT Standard Cipher (DSC) for encryption. The first attacks on DECT became known in 2008 [3]. Researchers demonstrated that encryption and even authentication could easily be switched off due to insecure DECT implementations that do not enforce them. Furthermore, the researchers observed that even when security is switched on, many devices use highly predictable random numbers thereby undermining the level of protection the DSC aims to achieve.

Since the initial findings, some handsets and base stations have been patched to enforce encryption and to use strong random numbers to mitigate the previous

S. Hong and T. Iwata (Eds.): FSE 2010, LNCS 6147, pp. 1–18, 2010.

attacks. Nonetheless, this paper demonstrates that even these improved devices can be attacked by exploiting weaknesses of the DSC.

The DECT Standard Cipher is an asynchronous stream cipher with low gate complexity that takes a 64 bit secret key and a 35 bit initialization vector, IV, to generate keystream. DSC is similar to GSM's A5/1 and was reverse-engineered from a DECT device using a combination of firmware probing and hardware reverse-engineering. The cipher, publicly disclosed for the first time[1] in this paper, is vulnerable against a clock guessing attack similar to the Ekdahl-Johansson attack [4] against A5/1. The attack – although it has a large data complexity – can be executed on a PC in hours and allows for passively sniffed voice and data connections to be decrypted. However, we were not able to carry over later improvements of the Ekdahl-Johansson attack [5,6] due to specific traits of A5/1 being used in them that are not present in the DSC.

DSC is stronger than A5/1 by statistical indicators such as the non-linearity of the round and filter function, key size and state size. However, DSC as used in DECT is initialized in less than half the number of rounds when compared to A5/1 in GSM. This underpins that the number of initialization rounds is a major security metric of stream ciphers.

The attack on DSC presented in this paper provides a trade-off between the number of available data samples and the time needed to calculate the secret state. When 2^{15} samples are available, the attack executes in less about 22 minutes on a 16 core Opteron machine clocked at 2.3GHz.

DSC and its use in DECT could be improved in several ways, most simply by increasing the number of initialization rounds. Incidentally, switching off encryption for the DECT control channel effectively increases the number of initialization rounds which as a side effect protects the data channel better. While this countermeasure does make our attack on data confidentiality more difficult, the DSC cipher – like many proprietary ciphers – is conceptually flawed and should not be used for security applications. The successor technology to DECT will hopefully include an open cipher that underwent extensive peer review to provide the appropriate level of confidentiality and authentication strength.

The paper is structured as follows: Section 2 gives a description of the DSC, Section 3 describes how the cipher was reverse-engineered and Section 4 shows attacks against the DSC. A high-performance implementation of DSC is described in Section 5; Section 6 discusses DSC's weaknesses and compares it to A5/1.

1.1 Notation

The internal state of DSC is represented as a 81 bit vector, $\mathsf{s} \in GF(2)^{81}$, comprised of the state of four linear feedback shift registers (LFSRs) and the memory bit of the output combiner.

[1] A partial description was given by the deDECTed.org project – of which the authors are members – at the 25th Chaos Communication Congress in Berlin in December 2008. At that point the output combiner and the key loading had not yet been reverse-engineered. This presentation also included the practical attacks described in [3].

Since state transitions are performed by linear operations, we will use matrices to describe them. The matrices in Table 1 represent the DSC operations:

Table 1. Matrices describing linear operations on the internal state

Matrix	Dimension	Description
C_1	81×81	single clock register R1
C_2	81×81	single clock register R2
C_3	81×81	single clock register R3
L	81×128	load key and IV into state
S	6×81	extract the first two leading bits from R1, R2, and R3

The output combiner $\mathcal{O}$ of DSC is a non-linear mapping, depending on the previous bit of output y and 6 bits of the state s.

The DSC round function that translates a state into the next round's state is a non-linear mapping. The pre-ciphering phase which consists of loading the secret key and initialization vector (IV) into the DSC registers and then applying the round function i times is denoted $\mathcal{D}_i$. The i initialization rounds are referred to as pre-ciphering steps.

2 Description of the DECT Standard Cipher

The DECT Standard Cipher (DSC) is an irregularly clocked combiner with memory. Its internal state is built from 4 Galois LFSRs R1, R2, R3, R4 of length 17, 19, 21 and 23 respectively as well as a single bit of memory y for the output combiner. The bits of the state of the LFSR Ri shall be denoted by $x_{i,j}$ with the lowest-most bit being $x_{i,0}$. The taps of R1 are located at bit positions 5, 16; the taps of R2 are at bit positions 2, 3, 12, 18; the taps of R3 at bit positions 1, 20; the taps of R4 are at bit positions 8, 22.

For each bit of output, register R4 is clocked three times whereas R1 to R3 are clocked either two or three times. The clocking decision is determined individually for each of the irregularly clocked registers. The decisions linearly depend on one of the three lowest bits of R4 and the middle bits of the other irregularly clocked registers. More specifically, the number of clocks c_i for each of the registers is calculated as follows:

$$
\begin{aligned}
c_1 &= 2 + (x_{4,0} \oplus x_{2,9} \oplus x_{3,10}) \\
c_2 &= 2 + (x_{4,1} \oplus x_{1,8} \oplus x_{3,10}) \\
c_3 &= 2 + (x_{4,2} \oplus x_{1,8} \oplus x_{2,9})
\end{aligned}
$$

2.1 The Output Combiner

The output combiner is a cubic function that involves the lowest-most two bits of the registers R1, R2 and R3 as well as the memory bit y:

$$\begin{aligned}\mathcal{O}((x_{1,0}, x_{1,1}, x_{2,0}, x_{2,1}, x_{3,0}, x_{3,1}), y) = {} & x_{1,1}x_{1,0}y \oplus x_{2,0}x_{1,1}x_{1,0} \oplus x_{1,1}y \oplus \\ & x_{2,1}x_{1,0}y \oplus x_{2,1}x_{2,0}x_{1,0} \oplus x_{3,0}y \oplus \\ & x_{3,0}x_{1,0}y \oplus x_{3,0}x_{2,0}x_{1,0} \oplus x_{3,1}y \oplus \\ & x_{1,1}x_{1,0} \oplus x_{2,0}x_{1,1} \oplus x_{3,1}x_{1,0} \oplus \\ & x_{2,1} \oplus x_{3,1}\end{aligned}$$

The output of the combiner function gives a keystream bit and is loaded into the memory bit for the next clock.

2.2 Key Loading and Initialization

Initially all registers and the memory bit are set to zero. The 35-bit IV is zero extended (most significant bits filled with zero) to 64 bits and concatenated with the 64 bit cipher key CK to form the session key K.

$$K = Z(IV)||CK$$

The bits of K are clocked into the most significant bit of all four registers, bit by bit, starting with the least significant bit. During the key loading each LFSR is clocked once after each bit. After the session key has been loaded, 40 pre-cipher rounds are performed. In these pre-cipher rounds, the irregular clock control is used but the output is discarded. If one or more registers have all bits set to zero after executing 11 rounds, the most significant bit of the corresponding registers is set to 1 before starting the next round.

3 Reverse-Engineering the DSC from Hardware

We did not find any software implementations of DSC. Instead our starting point was a patent [7] describing the general structure of the DSC. From this document we learn that DSC is an LFSR-based design together with the lengths of the individual registers. Furthermore the patent discloses that the cipher has an output combiner with a single bit of memory, irregular 2-3 clocking and the number of initialization rounds. On the other hand the tap positions of the LFSRs, the clocking functions, the combiner function as well as the exact key loading routine are not described in this patent. The rule that after 11 initialization rounds a check had to be performed to make sure that no register is zero at that point is also stated in the patent.

Luckily, for the National Semiconductors SC14xxx DECT chipset that is used by the deDECTed.org project, we found instructions that allow to load and store an abitrary internal state of the stream cipher. Moreover, the stream cipher can be clocked in two modes: a regular clocking of the the LFSRs for key loading and a second mode clocking irregularly as specified by the clocking functions, generating output. However we are not able to directly capture these output bits.

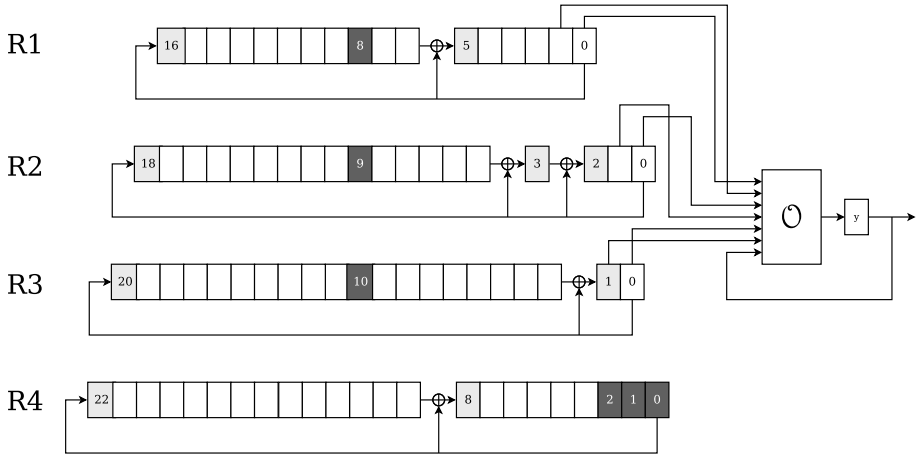

Fig. 1. The DSC keystream generator with LFSRs in Galois configuration. Bit positions that are inverted (white on black) are used in clocking decisions.

To reverse-engineer the unspecified details of the cipher we proceed as follows: Using the first mode allows us to determine the tap positions of the LFSRs. After that, we are able to determine the clocking functions in the second mode by loading a random vector of low Hamming weight into the internal state and observing how single-bit changes affect the clocking decisions.

The most elaborate part to reverse-engineer is the output combiner function. To do this, we set up one machine with a modified firmware to send out frames containing zero-stuffed payloads. Another machine acting as the receiving side then "decrypts" these using a chosen internal state (no key setup), yielding keystream. Starting from random states, we sequentially flip single bit positions of the state and inspect the first bit to see whether the bit flip affected the output. If the output remains constant for a large number of random states, we assume that the flipped bit is not used in the output combiner. Having identified the bits that indeed are fed into the combiner, we recover the combiner function by using multivariate interpolation for a number of keystreams.

Finally we determine the correct key loading by systematically trying different bit and byte-orders for both key and IV combined with both different orders of key and IV.

In parallel to having done the above, we also reverse-engineered the DSC cipher including its output combiner from silicon applying the techniques previously used to discover the Crypto-1 function [8].

4 Attacking the DSC

For this section, we will assume that an adversary has access to a list of DSC keystreams with matching initialization vectors, all of which were generated under the same secret key. A tuple of keystream and IV is referred to as a *session*.

We will use clock-guessing techniques very similar to the Ekdahl-Johansson attack against A5/1 [5], but adapted to the case of a non-linear output combiner. Further improvements of this attack discussed in [5,6] seem to be too specific to the A5/1 structure.

4.1 Simple Clock Guessing

Despite its relative large state and non-linearity, DSC is easily broken because of one major design flaw: the small number of pre-ciphering rounds makes clock guessing easy. After the key loading, there are only 40 clocking decisions made, compared to 100 clocking decisions for A5/1.

If an internal state for DSC is randomly chosen from a uniform distribution of all states, every irregularly clocked register clocks twice with 50% probability or three times with 50% probability. We assume for now that the probability that one register is clocked twice is independent of the clocking decision of the other irregularly clocked registers. The probability that one register is clocked k times during the pre-ciphering-phase is:

$$\binom{40}{k-80}2^{-40}$$

and the probability that a register has been clocked k times after i bits of output is:

$$\binom{40+i}{k-(80+2i)}2^{-(40+i)}$$

The total number of clocks per register after i bits of output is distributed according to a shifted binomial distribution with mode:

$$\left\lfloor \frac{i+1}{2} \right\rfloor + 2i + 100$$

In general, let

$$\mathsf{D}^{i,j,k} = \mathsf{S} \times \mathsf{C}_1^i \times \mathsf{C}_2^j \times \mathsf{C}_3^k \times \mathsf{L} \times (\mathsf{key}, \mathsf{iv})$$

be the state of the six bits of the registers which generate the output, after key and iv have been loaded, and register R1 has been clocked i, register R2 has been clocked j, and register R3 has been clocked k times.

The attack focuses on the internal DSC state from which the second bit of output is produced. An attacker who has observed the first bit of output knows the state z_0 of the memory bit of the output combiner. The second bit of output depends on 6 bits of the registers R1, R2, and R3. With a probability of

$$p = \left(\binom{41}{21}2^{-41}\right)^3 \approx 2^{-9.09}$$

all of these irregularly clocked registers will be clocked exactly 103 times before the second bit of output z_1 is produced and we have

$$\mathsf{D}^{103,103,103}(\mathsf{key}, \mathsf{iv})) = \mathsf{S} \times \mathcal{D}_{41}(\mathsf{key}, \mathsf{iv})$$

with probability 1. If the number of clocks per register is different, this equation will hold by chance with a probability $\frac{1}{64}$. Therefore, we have

$$\text{Prob}\left(\mathsf{D}^{103,103,103}(\mathsf{key},\mathsf{iv}) = \mathsf{S} \times \mathcal{D}_{41}(\mathsf{key},\mathsf{iv})\right) = p + \frac{1-p}{64} \approx 2^{-5.84}$$

Based on this guess, six affine-linear equations for an unknown key can be derived given that sufficiently many keystreams (about 2^{18}) are available. For every IV iv, the attacker computes

$$\mathsf{I}^{103,103,103} = \mathsf{D}^{103,103,103}(0,\mathsf{iv})$$

He then checks for every possible state s of the six output bits whether $\mathcal{O}(s, z_0) = z_1$ holds. If so, this is an indication that

$$\mathsf{D}^{103,103,103}(\mathsf{key},0) = s + \mathsf{I}^{103,103,103}$$

holds. After having processed all available keystreams, the attacker may assume the most frequent value for $\mathsf{D}^{103,103,103}(\mathsf{key},0)$ to be correct. Using these six affine-linear equations allows an attacker to recover the correct key by trying only 2^{58} instead of 2^{64} possible keys. This basic attack however is still too time-consuming to be practical on a single PC.

4.2 Breaking DSC on a PC

We can refine the basic attack principle to give us a much faster attack that allows us to practically recover a DSC key on a PC.

Note that the attack scope can be extended to different assumptions for the number of clockings for the registers R1, R2 and R3. In the basic attack, we use the mode of the distribution of the number of clock of all registers. If the total number of clocking decisions is odd, another set of clockings with the same success rate always exists. For example, if the attacker assumes that R1 and R2 have been clocked 103 times as in the previous subsection, but R3 has been clocked 102 times instead of 103 times, the previous attack works with the same computational effort and success rate. In total, there are 8 possible assumptions about the number of clocks with the same success rate as the previous attack.

However, these different assumptions share many equations for the key. An attacker will only obtain nine different affine-linear equations for the key using these eight assumptions (compared to six equations for a single assumption).

Extending the attack scope further, – i.e., assuming that R1 has been clocked 101 times, and R2 and R3 have been clocked 102 times – increases the success rate of the attack but at an even smaller incremental gain per additional assumption.

Another way to broaden the attack is to focus on different keystream bits. The basic attack only uses the first two bits of the output, z_0 and z_1. Instead of guessing how many times the registers have been clocked before producing the z_1, one could guess how many times the registers have been clocked before z_2 is produced. For example, an attacker can try using z_1 and z_2 of the output and

guess that R1, R2, and R3 have been clocked exactly 105 times. The resulting correlation will have the same success rate as the one from the basic attack. Using multiple output bits for a single clocking triplet is possible.

Combining these two time-success trade-offs, we developed a more advanced key-recovery attack on the DSC that merely requires hours of computation on a PC given enough keystreams. We chose a clocking interval C = [102, 137], and generated all $35^3 = 42875$ possible approximations with the number of clocks of R1 to R3 in C. We introduce new variables $x_{i,j}^{(t)}$ for the state of bit j of Register Ri after it has been clocked t times. Assuming that Ri has been clocked t times for an approximation gives us information about $x_{i,0}^{(t)}$ and $x_{i,1}^{(t)}$. In total, a clocking-interval of length l gives us information about $6l$ variables (3 registers, 2 variables per clocking amount). However, $x_{i,1}^{(t)} = x_{i,0}^{(t+1)}$ holds for all registers Ri, because $x_{i,1}^{(t)}$ is just shifted to $x_{i,0}^{(t+1)}$ with the next clock of Ri. Choosing a different feedback polynomial with a feedback position between the two bits contributing to the output combiner would destroy this structure. However for DSC, all feedback polynomials don't have a feedback position here. Effectively, this gives us information about $3(l+1)$ variables for a clocking interval of length l. We will always use $x_{i,0}^{(t+1)}$ instead of $x_{i,1}^{(t)}$ for the rest of this paper.

There are also linear relations between these variables. For example $x_{1,5}^{(t+1)} = x_{1,6}^{(t)} \oplus x_{1,0}^{(t)}$ holds. In general, having determined a consecutive sequence of variables $x_{i,0}^{(t)}, x_{i,0}^{(t+1)}, \ldots$ for a register Ri, is equivalent to the output sequence of Ri. If more variables than the length of Ri have been determined, one might use these linear relations to check if a given assignment is feasible. However, we did not use this in our attack.

The success rate that register R1 is clocked i times, register R2 is clocked j times and register R3 is clocked k times after l bits of output have been produced is:

$$p_{i,j,k,l} = \binom{40+l}{i-(80+2l)}\binom{40+l}{j-(80+2l)}\binom{40+l}{k-(80+2l)}2^{-(40+l)3}$$

In theory, one could use all available bits of keystream for which the correlation has better than zero success rate, however after 19 bits of keystream, all of these correlations have negligible success probability. For example the probability that all registers have been clocked 137 times (the end of our clocking-interval) for the 19th bit of output is below 2^{-26}.

As in the basic attack, we evaluate all correlations separately and create a frequency table for every correlation. Following the ideas of Maximov et al.[5] we add the log-likelihood ratio $\ln \frac{p}{1-p}$ for $\mathsf{key} = s + \mathsf{iv}$ to every entry in the table, with

$$p = \sum_l p_{i,j,k,l} * [\mathcal{O}(s, z_{l-1}) = z_l] + \frac{1}{2}\left(1 - \sum_l p_{i,j,k,l}\right)$$

Here $[\mathcal{O}(s, z_{l-1}) = z_l] = 1$, if $\mathcal{O}(s, z_{l-1}) = z_l$, 0 otherwise.

Instead of writing the equations in all correlations as a linear combination of key bits, we now write all equations in the form $x_{\{1,2,3\},0}^{(i)} = \{0, 1\}$.

Taking the entry with the highest probability from the frequency table of every approximations, we obtain $42875 * 6 = 257250$ equations with a given probability. (Every approximation (42875) gives us information about the value of 6 state-variables. In total, these state-variables can have $2^6 = 64$ possible values, the value with the highest probability in the frequency table is most likely. We use the number of (weighted votes) for the top entry as an extend p_i how likely these equations are correct.) For every variable x, we take all the equations of the form $x = b_i, b_i \in \{0, 1\}$ with extend p_i and compute $s_x = \sum_i (2b_i - 1) * p_i$ and assume that $x = 0$, if $s_x < 0$, $x = 1$ otherwise.

Combining all equations to a single equation system gives 108 equations each of which depends only on a single variable and a corresponding probability p_v that this equation is correct. We sort these equations according to $|p_v|$, rewrite all variables to key bits, and add them in order to a new linear equation system for the key bits. If adding a equation would make the resulting system unsolvable, we skip that equation. If enough linearly independent variables (for example 30) have been added to the system, we stop the process.

We then iterate through all solutions to this system, and check every solution if it is the correct key, by comparing it to some sample keystreams.

We created a proof-of-concept implementation of this attack written in Java. Processing all available keystreams and generating a linear equation system takes about 20 minutes using a SUN X4440 using 4 *Quad-Core AMD Opteron(tm) Processor 8356* running at 2.3 GHz. The main workload here is the generation of all the frequency tables for all approximations. The post processing and the generation of the final equation system is negligible. We think that this time can be reduced to a few minutes using parallel computation and a more efficient implementation. For the time for the final search of the correct key, see Section 5.

The success rate of this attack depends on the number of available keystreams and the number of equations in the final equation system for the keybits. Using more equations makes the final search for the correct key faster, but increases the probability of having at least one incorrect equation in the system which makes the attack fail. If i equations are used in the final system, one still needs to search through at most 2^{64-i} different keys to find the correct key (assuming the equation system is correct).

Using 30 equations in the final system (one still needs to check at most 2^{34} different keys), the attack was successful in 48 out of 100 simulations with 32768 different keystreams available. Using only 16384 keystreams, the success rate dropped down to 1%. With 49152 keystreams, the attack was successful in 95% of all simulations. If only 8 equations should be used, the attack had a success rate of 8% using just 8192 keystreams. However an adversary would need to conduct a final search for the key over 2^{56} different keys, which is roughly equivalent to a brute force attack against DES.

4.3 Keystream Recovery

To break the DSC stream cipher, keystream needs to be recovered from the encrypted frames, which is only possible when the user data is known or can

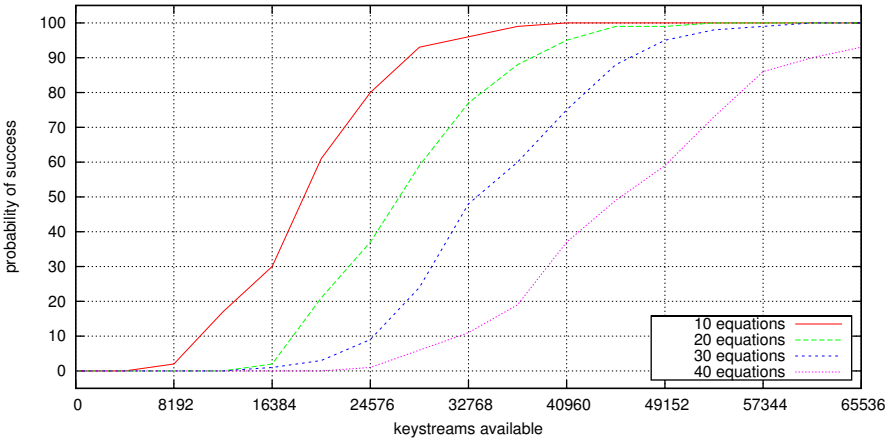

Fig. 2. Success rate of the attack

be guessed. Known user data is regularly sent over DECT's control channel (C-channel). The C-channel messages (e.g., for a button press) share a common structure in which the majority of the first 40 bits stays constant. There are at most 50 C-channel packets sent per second which provides an upper bound on the number of known keystream segments from the C-channel. Especially in newer phones, the C-channel is extensively used for status updates including RSS feeds and other data communication which opens the possibility that a significant number of known keystream can be gathered.

Keystreams can also be collected from the voice channel (B-field), but assumptions have to be made about the voice being transmitted (i.e., segments of silence). Even when these assumptions do not hold in all cases, the data is still usable in the attacks outlined below as they are error-resilient. More information can be found in appendix A.

4.4 Extending the Attack to B-Field Data

Thus far we have assumed the adversary to have access to the first bits of output of DSC after pre-ciphering. However, these bits are only used to encrypt the C-channel data in DECT. If C-channel data is not frequently used in a conversation, the adversary is unable to recover a sufficient number of keystreams using the techniques previously described.

Henceforth, we adapt our attack to also work when the first 40 bits of keystream are not available. To achieve this, we need to change the clocking interval from $[102, 137]$ to $[204, 239]$. We then use 21 bits of the keystream starting from bit 41. The best approximation which exists is to assume that every register has been clocked 202 times when the second bit for the B-field is produced. This happens with probability $2^{-10.527}$, instead of $2^{-9.0915}$ for the best approximation for the C-channel bits.

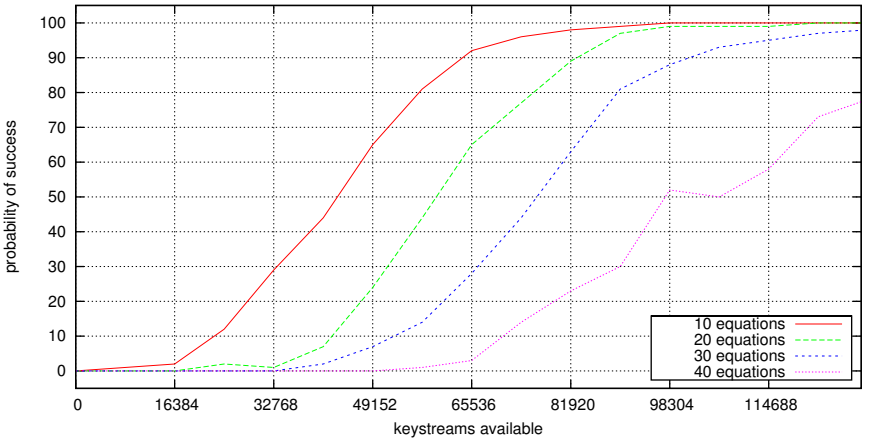

Fig. 3. Success rate of the B-field attack

As expected, the number of keystreams required for the same success rate is increased by factor of 2-3. To make the attack work with a success rate of 50%, the attack requires 75,000 keystreams. Again we conducted 100 simulations to experimentally verify the success rate and to generate the plot in figure 3.

However, B-fields are sent 100 times per second from FP to PP while a call is in progress. This allows to recover the corresponding keystreams in less than 13 minutes if a predictable plaintext pattern is used in the B-field.

5 High-Performance DSC

The DSC is optimized for hardware implementations where LFSRs can be implemented in a small number of logic gates. To minimize the complexity of our attack implementation, we did not facilitate FPGAs or even build custom ASICs for DSC computations. Instead we optimized implementations for an x64 CPU, a NVidia CUDA graphics card–both of which are commonly found in home computer–and a cell processor. Optimizations we applied to accelerate the attack include bit slicing and the use of bit vectors. The combination of these tweaks resulted in a 25x speed-up on Intel CPUs when compared to a straightforward DSC implementation. Performance figures for components in a standard gaming PC (CPU, NVidia graphics adapter) and a PlayStation 3 (Cell processor) are provided in Table 2, including the search time for the C-channel attack searching through 2^{34} keys.

Our optimizations have been implemented for Intel CPUs (128bit SSE), the Cell processor in PlayStations (128bit wide SIMD units in both the SPEs and the PPE) and to CUDA GPUs where only 64 bit wide general purpose registers are available. However, due to the large number of compute kernels, a high-end CUDA graphics card is almost one order of magnitude faster than a high-end Intel CPU for our attack.

Table 2. Efficiency of brute-forcing large numbers of DSC keys on different architectures

Compute Node	Keys tried per second	Attack time for final search
Intel 2 GHz CPU (2 cores)	24 million	716s
Cell processor	25 million	687s
CUDA GTX260	148 million	116s

Since the problem is parallelizable without dependencies, utilization of multicore systems comes without penalty allowing the key cracking to be distributed over a large number of CPU, cell, and CUDA nodes.

Bit slicing. In the optimized implementation, the 81 bit state of DSC is stored in 81 registers, each register holding 128 bits for 128 DSC engines. With only 3 xor instructions the clocking decision for R1 can be made – for 128 DSC states at once. A first implementation for an Intel Core 2 Duo at 2Ghz can verify 12 million key candidates per second per core, including extraction of the 64 bit key from the equation system, the 64 bit key setup procedure to clock the key into the DSC state, 40 clockings of DSC in the pre-cipher phase and the generation and comparison of 8 keystream bits. The amortized simulator processor clocks to produce a bit of DSC output is about 2.

The bit slice implementation has the drawback that there is no efficient way to shift a register, so for a simple one bit shift of R1, 17 locations of memory have to be copied in the ring buffer. Fortunately when the clocking is regular, copying is not necessary and single combined head and tail pointer can be incremented and decremented to facilitate the shift operation. This optimization is usable during key setup and for R4 during all DSC stages.

Bit vectors. The extraction of the candidate keys from the equation system is further optimized through the use of bit vectors, so 64 operations are needed to produce 128 64 bit long keys, by first building a template for key values encoding the information of any set of 128 consecutive candidate keys and using this template to produce 128 keys anywhere in the key-space with linear complexity over the number of bits (64).

CUDA tweaks. High end graphics cards for computer gaming can be expected to perform at least 10 times faster then a single CPU. One such GPU has 240 ALUs clocked at about 1.2Ghz. The slower clock speed and the smaller register size of 64 bits each halve the effect of the larger number of cores. Furthermore CPU features such as superscalar execution, branch prediction and out of order execution narrow the gap further. At about three times the power consumption, a high-end CUDA GPU still executes about ten times as fast as a Intel CPU making it the preferred host for our attack.

6 DSC Weaknesses and Mitigations

The DSC cipher is vulnerable against the attack described in this paper because it does not accumulate enough non-linearity before producing the first keystreams. Our attack DSC exploits the fact that the cipher can be expressed in relatively simple equations that hold true with non-negligible probability. These equations can be generated because three weaknesses come together in DSC:

- A round function with a low level of non-linearity
- An insufficiently small number of rounds before the first key stream bit is produced
- Access to keystreams through known plaintext in the C-channel

The latter two properties make DSC much weaker than the related A5/1 stream cipher used in GSM[2]. Attacks on A5/1 still require more keystream than can be inferred from GSM packets or extensive precomputations for time-memory trade-offs [9].

In other dimensions of statistical strength, the DSC cipher is stronger than A5/1, again emphasizing how serious the above mentioned weaknesses are.

Table 3. Comparing A5/1 against the DSC

	A5/1	DSC
number of registers	3	4
irregularly clocked registers	3	3
internal state in bits	64	81
output combiner	linear	non-linear
bits used for output	3	7
bits used for clocking	3	6
clocking decision	0/1	2/3
clocks per register until first bit of output	0-100	80-120
average clocks of registers until first bit of output	75	100
pre-cipher rounds	100	40

The larger internal state makes practical time-memory tradeoffs infeasible for DSC. Since more bits are used in the output combiner, the sampling of special states [10] is much harder for DSC than for A5/1. At the same time, the non-linearity of the output combiner in DSC improves its resilience against divide and conquer strategies. DSC has a register which only affects the clocking control and doesn't directly generate the keystream. In A5/1 every register affects the output directly. Differential attacks against A5/1 [11] use that fact that this cipher does not always clock all registers; the DSC clocks every register at least two times after each bit of output.

[2] A5/1 and DSC were standardized by the same organization, A5/1 in 1987 and DSC in 1992.

The attack outlined in this paper is enabled through available keystream and the small number of clocking rounds. Short-term countermeasures to mitigate the risk imposed by this particular attack include:

- frequent re-keying to prevent an attack from collecting sufficiently many keystreams.
- switching off encryption of the C-channel (which might lead to privacy concerns over dialed numbers etc.);

Both measures can be deployed to many existing base stations and handsets through firmware updates.

However, it is our belief that the current DECT standard includes too many vulnerabilities to be made secure through quick fixes. Also, DECT with DSC does not provide sufficient security for its intended uses. To provide this level of protection, a strong peer-reviewed cipher and security protocol are needed in the next version of the DECT standard.

7 Conclusions

Cryptographic functions can be reverse-engineered from hardware devices and need to hold up to security analyses when they are disclosed. The widely-used DSC cipher in DECT cordless phones did not hold up to the test of a curious review.

While the attack presented in this paper does rely on strong assumptions on the availability of keystreams, other attacks will very likely further degrade the security level of DSC. We believe that the DSC cipher as used in DECT is not sufficient to protect the confidentiality of personal conversations, network traffic, and credit card information. We propose the DECT standard to be amended by a strong, peer-reviewed cipher in order to provide sufficient protection for its users.

The fact that the DSC cipher has not undergone public review, despite being deployed in hundreds of millions of locations, raises the question of why not more proprietary security algorithms have been reverse-engineered. The techniques used for reversing DSC from a firmware and a chip implementation can certainly be further generalized. This opens interesting research avenues; research targets are aplenty as new proprietary ciphers in embedded applications such as car buses and RFID chips are constantly being created.

Acknowledgements. This paper builds on software and firmware created within the deDECTed.org project. Moreover, a DECT kernel stack for Linux written by Patrick McHardy was used in the reverse engineering process. We would like to thank Andreas Schuler who helped us writing firmware for the SC14421 to reverse-engineer the cipher, Sascha Krissler for implementing the DSC on CUDA and Starbug for his silicon reverse engineering work. Especially we would like to thank the anonymous reviewers, who had very valuable ideas for improvements of the attack. In particular they pointed us to a publication [4] by Ekdahl and Johansson which shows an interesting attack against A5/1; this helped us to significantly improve our attack.

Open probems. Our attacks against DSC should be seen as a starting point. Although we were not able to carry over the improvements for the Ekdahl-Johansson attack against A5/1 to the DSC we challenge other researchers to give it a try.

The statistical methods we used to generate our equation systems certainly can be improved. Thus far, all approximations in the clocking interval have been used. A significant amount of CPU time in the attack is used for processing all keystreams with all approximations. A more sophisticated method could select a subset of the approximations to get an improved running time.

Moreover, it is an interesting problem to see whether the number of keystreams required can be reduced by taking the linear equations from the feedback polynomials into account.

The most interesting problem however to us is to find an attack against the DSC that works with a very small number of keystreams. Due to the structure of our attack we do not believe that it can be adapted to this scenario. At the same time, other attacks with a low data complexity that work against A5/1 cannot be carried over to DSC due to the larger internal state size.

References

1. MZA Telecoms & IT Analysts: Global cordless phone market. Press Release (August 2009)
2. DECT Forum: Positioning of DECT in relation to other radio access technologies. Report (June 2002)
3. Lucks, S., Schuler, A., Tews, E., Weinmann, R.P., Wenzel, M.: Attacks on the DECT authentication mechanisms. In: Fischlin, M. (ed.) RSA Conference 2009. LNCS, vol. 5473, pp. 48–65. Springer, Heidelberg (2009)
4. Ekdahl, P., Johansson, T.: Another attack on A5/1. IEEE Transactions on Information Theory 49(1), 284–289 (2003)
5. Maximov, A., Johansson, T., Babbage, S.: An improved correlation attack on A5/1. In: Handschuh, H., Hasan, M.A. (eds.) SAC 2004. LNCS, vol. 3357, pp. 1–18. Springer, Heidelberg (2004)
6. Barkan, E., Biham, E.: Conditional estimators: An effective attack on A5/1. In: Preneel, B., Tavares, S. (eds.) SAC 2005. LNCS, vol. 3897, pp. 1–19. Springer, Heidelberg (2006)
7. Alcatel: Data ciphering device. U.S. Patent 5,608,802 (1994)
8. Nohl, K., Evans, D., Starbug, Plötz, H.: Reverse-engineering a cryptographic RFID tag. In: van Oorschot, P.C. (ed.) USENIX Security Symposium 2008, USENIX Association, pp. 185–194 (2008)
9. Barkan, E., Biham, E., Keller, N.: Instant ciphertext-only cryptanalysis of GSM encrypted communication. Journal of Cryptology 21(3), 392–429 (2008)
10. Biryukov, A., Shamir, A., Wagner, D.: Real time cryptanalysis of A5/1 on a PC. In: Schneier, B. (ed.) FSE 2000. LNCS, vol. 1978, pp. 1–18. Springer, Heidelberg (2001)
11. Biham, E., Dunkelman, O.: Differential cryptanalysis in stream ciphers. Cryptology ePrint Archive, Report 2007/218 (2007), `http://eprint.iacr.org/2007/218`

A Technical Background on DECT for Keystream Recovery

DECT divides carrier frequencies into multiple timeslots. An interval of 10 ms is divided into 24 timeslots of equal length. Connections in DECT are always between a base station – in DECT terminology an FP (Fixed Part) – and a handset, called PP (Portable Part). An FP typically uses a single timeslot $i \in \{0, \ldots, 11\}$ to transmit a full frame to the PP; the PP then responds in a single timeslot $i + 12$ with a frame. DECT supports multiple frame formats with different modulations; some of which use half a time slot or two consecutive timeslots.

A single DECT full frame using GFSK modulation consists of a 16 bit static preamble, a 64 bit A-Field, a 320 bit B-Field, and two 4 bit checksums. The A-Field can transport data for the C-,M-,N-,P-, or Q-Channel. If an A-Field is used to transport C-channel messages, only 40 bits of the A-Field contain C-channel data, the rest is used for header-bits.

If encryption is active, the DECT Standard Cipher (DSC) generates 720 consecutive bits of keystream for every frame exchange. The output is divided into two keystream segments (KSS), the first 360 bit KSS is used to encrypt the frame sent from the FP to the PP, the second KSS is used to encrypt the frame sent from PP to FP.

If C-channel data is present in the A-Field, the first 40 bits of the KSS are XORed with these bits, otherwise they are discarded. The remaining 320 bits of the KSS are XORed with the B-Field.

Keystream can only be recovered for cryptanalysis from frames where the plaintext is known. Two examples where plaintext is guessable are:

- Some phones display a counter with the duration of the current call in the *hh:mm:ss* format. This counter is usually implemented on the base station. The display of the phone is updated once per second by the base station with the next counter value. We observed a single C-channel message being split into 5 frames. Intercepting these messages recovers 5 different keystreams per second for which most of the first 40 bits are known.
- When (perfect) silence is transmitted the G.721 audio codec produces plaintext of only ones. Some applications like voice mailboxes transmit silence in one direction after the greeting message. This can be used to recover up to 100 frames per second with 320 known bits known. The first 40 bits are only used for the C-channel and cannot be recovered using this method.

B An implementation of the DSC in C

```
#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

#define R1_LEN                  17
#define R2_LEN                  19
#define R3_LEN                  21
#define R4_LEN                  23

#define R1_MASK                 0x010020 /* tap bits: 5, 16 */
#define R2_MASK                 0x04100C /* tap bits: 2, 3, 12, 18 */
#define R3_MASK                 0x100002 /* tap bits: 1, 20 */
#define R4_MASK                 0x400100 /* tap bits: 8, 22 */

#define R1_CLKBIT               8
#define R2_CLKBIT               9
#define R3_CLKBIT               10
#define R1_R4_CLKBIT            0
#define R2_R4_CLKBIT            1
#define R3_R4_CLKBIT            2

#define OUTPUT_LEN              90 /* 720 bits */
#define TESTBIT(R, n)           (((R) & (1 << (n))) != 0)

uint32_t clock(uint32_t lfsr, uint32_t mask) {
  return (lfsr >> 1) ^ (-(lfsr&1)&mask);
}

uint32_t combine(uint32_t c, uint32_t r1, uint32_t r2, uint32_t r3) {
  uint32_t x10, x11, x20, x21, x30, x31;

  x10 = r1 & 1;
  x11 = (r1 >> 1) & 1;
  x20 = r2 & 1;
  x21 = (r2 >> 1) & 1;
  x30 = r3 & 1;
  x31 = (r3 >> 1) & 1;

  return ((x11&x10&c) ^ (x20&x11&x10) ^ (x21&x10&c) ^ (x21&x20&x10) ^
          (x30&x10&c) ^ (x30&x20&x10) ^ (x11&c) ^ (x11&x10) ^ (x20&x11) ^
          (x30&c) ^ (x31&c) ^ (x31&x10) ^ (x21) ^ (x31));
}

void dsc_keystream(uint8_t *key, uint32_t iv, uint8_t *output) {

  uint8_t input[16];
  uint32_t R1, R2, R3, R4, N1, N2, N3, COMB;
  int i, keybit;

  memset(output, 0, OUTPUT_LEN);

  input[0] = iv&0xff;
  input[1] = (iv>>8)&0xff;
  input[2] = (iv>>16)&0xff;
  for (i = 3; i < 8; i++) {
        input[i] = 0;
  }
  for (i = 0; i < 8; i++) {
        input[i+8] = key[i];
  }

  R1 = R2 = R3 = R4 = COMB = 0;
```

```
  /* load IV || KEY */
  for (i = 0; i < 128; i++) {
    keybit = (input[i/8] >> ((i)&7)) & 1;
    R1 = clock(R1, R1_MASK) ^ (keybit<<(R1_LEN-1));
    R2 = clock(R2, R2_MASK) ^ (keybit<<(R2_LEN-1));
    R3 = clock(R3, R3_MASK) ^ (keybit<<(R3_LEN-1));
    R4 = clock(R4, R4_MASK) ^ (keybit<<(R4_LEN-1));
  }

  for (i = 0; i < 40 + (OUTPUT_LEN*8); i++) {
    /* check whether any registers are zero after 11 pre-ciphering steps.
     * if a register is all-zero after 11 steps, set input bit to one
     * (see U.S. patent 5608802)
     */
    if (i == 11) {
      if (!R1) R1 ^= (1<<(R1_LEN-1));
      if (!R2) R2 ^= (1<<(R2_LEN-1));
      if (!R3) R3 ^= (1<<(R3_LEN-1));
      if (!R4) R4 ^= (1<<(R4_LEN-1));
    }

    N1 = R1;
    N2 = R2;
    N3 = R3;
    COMB = combine(COMB, R1, R2, R3);

    if (TESTBIT(R2, R2_CLKBIT) ^ TESTBIT(R3, R3_CLKBIT) ^
        TESTBIT(R4, R1_R4_CLKBIT))
      N1 = clock(R1, R1_MASK);
    if (TESTBIT(R1, R1_CLKBIT) ^ TESTBIT(R3, R3_CLKBIT) ^
        TESTBIT(R4, R2_R4_CLKBIT))
      N2 = clock(R2, R2_MASK);
    if (TESTBIT(R1, R1_CLKBIT) ^ TESTBIT(R2, R2_CLKBIT) ^
        TESTBIT(R4, R3_R4_CLKBIT))
      N3 = clock(R3, R3_MASK);

    R1 = clock(clock(N1, R1_MASK), R1_MASK);
    R2 = clock(clock(N2, R2_MASK), R2_MASK);
    R3 = clock(clock(N3, R3_MASK), R3_MASK);

    R4 = clock(clock(clock(R4, R4_MASK), R4_MASK), R4_MASK);

    if(i >= 40) {
      output[(i-40)/8] |= ((COMB) << ( 7-((i-40)&7)));
    }
  }
}

int main(int argc, char**argv) {
  uint8_t key[8];
  uint8_t output[OUTPUT_LEN];

  if (argc != 2) {
    fprintf(stderr, "usage:␣%s␣iv\n", argv[0]);
    exit(1);
  }

  if (read(STDIN_FILENO, key, 8) < 8) {
    fprintf(stderr, "short␣read\n");
    exit(1);
  }

  dsc_keystream(key, atoi(argv[1]), output);

  write(STDOUT_FILENO, output, 2048);
  return 0;
}
```